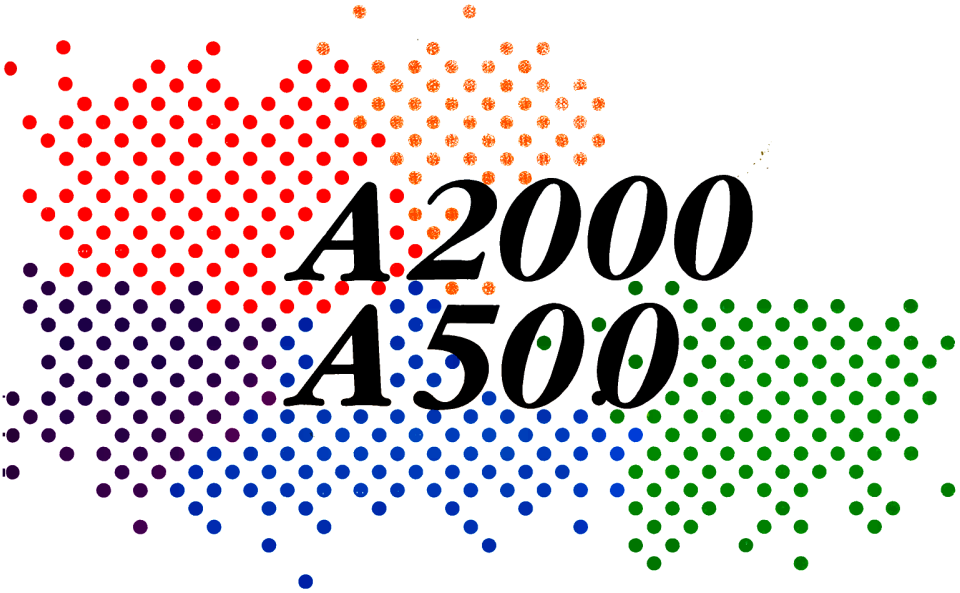


Amiga Basic
Deutsch

Commodore

A2000
A500

Amiga Basic
Deutsch

Commodore

Part.-Nr. 380798-02
1 2 3 4 90 B

Inhaltsverzeichnis

1. Amiga Basic – Eine Übersicht 1– 1
1.1 Spezielle Eigenschaften von Amiga Basic 1– 3
1.2 Leistungsfähige Sprach–Besonderheiten 1– 4
1.3 Unterstützung der besonderen Amiga–Möglichkeiten 1– 6

2. Mit Amiga Basic vertraut werden 2– 1
2.1 Ein praktisches Beispiel 2– 3
2.2 Beenden von Amiga Basic und Rückkehr zum Arbeitstisch (Workbench) 2– 19

3. Arbeiten mit Amiga Basic 3– 1
3.1 Grundlegende Funktionen 3– 1
3.2 Betriebsarten von Amiga Basic 3– 3
3.3 Der Amiga Basic–Bildschirm 3– 4
3.4 Die Menü–Leiste und Tastaturabkürzungen für das Menü 3– 7

4. Edieren und Testen von Amiga Basic–Programmen 4– 1
4.1 Programm–Edierung 4– 1
4.2 Testen von Amiga Basic–Programmen 4– 4
4.3 Programmtest mit dem Ausgabefenster 4– 6

5. Arbeiten mit Dateien und Geräten 5– 1
5.1 Allgemeine Geräte–Ein–/Ausgabe 5– 1
5.2 Dateibenennungs–Konventionen 5– 4
5.3 Das Arbeiten mit Dateien 5– 5
5.4 Datendateien für sequentiellen und direkten Zugriff 5– 7
5.5 Datenaustausch zwischen Amiga Basic und Textprogrammen 5– 21

6. Besonderheiten von Amiga Basic 6– 1
6.1 Unterprogramme 6– 1
6.2 Aufruf von Maschinensprache–Unterprogrammen 6– 9
6.3 Aufruf von Bibliotheksroutinen 6– 12
6.4 Verarbeitung von Unterbrechungs–Ereignissen 6– 15
6.5 Speicherverwaltung 6– 17

7. Erzeugen bewegter Bilder mit dem Objekt–Editor 7– 1
7.1 Übersicht 7– 1
7.2 Das Editor–Fenster 7– 2
7.3 Die Editor–Menüs 7– 4
7.4 Eine Bemerkung über "Bobs" und "Sprites" 7– 5
7.5 Wie man Objekte erstellt 7– 6

8. Die Elemente der Amiga Basic–Sprache 8– 1
8.1 Der Zeichensatz 8– 1
8.2 Die Amiga Basic–Programmzeile 8– 3
8.3 Definition von Sprungmarken 8– 4
8.4 Konstanten 8– 5
8.5 Variablen 8– 8
8.6 Numerische Ausdrücke und Operatoren 8– 12
8.7 Operatoren bei Zeichenketten 8– 18

9. Alphabetisches Verzeichnis aller Befehle, Anweisungen, Funktionen und Systemvariablen 9– 1
9.1 Struktur von Amiga Basic 9– 1
9.2 Struktur und Syntax der Beschreibungen 9– 2
9.3 Beschreibung der einzelnen Befehle, Anweisungen, Funktionen und Variablen 9– 3

ABS–Funktion 9– 21
AREA–Anweisung 9– 22
AREAFILL–Anweisung 9– 23
ASC–Funktion 9– 24
ATN–Funktion 9– 25

BEEP–Anweisung 9– 26
BREAK–Anweisung 9– 27

CALL–Anweisung 9– 28
CDBL–Funktion 9– 31
CHAIN–Anweisung 9– 32
CHDIR–Befehl 9– 34
CHR$–Funktion 9– 36
CINT–Funktion 9– 37
CIRCLE–Anweisung 9– 38
CLEAR–Befehl 9– 40
CLNG–Funktion 9– 42
CLOSE–Anweisung 9– 43
CLS–Anweisung 9– 44

COLLISION–Anweisung ... 9– 45
COLLISION–Funktion ... 9– 46
COLOR–Anweisung ... 9– 48
COMMON–Anweisung ... 9– 49
CONT–Befehl ... 9– 50
COS–Funktion ... 9– 51
CSNG–Funktion ... 9– 52
CSRLIN–Systemvariable ... 9– 53
CVI–, CVL–, CVS–, CVD–Funktionen ... 9– 54

DATA–Anweisung ... 9– 55
DATE$–Systemvariable ... 9– 57
DECLARE FUNCTION–Anweisung ... 9– 58
DEF FN–Anweisung ... 9– 59
DEF *Typ*–Anweisungen ... 9– 61
DELETE–Befehl ... 9– 63
DIM–Anweisung ... 9– 64

END–Anweisung ... 9– 66
END SUB–Anweisung ... 9– 67
EOF–Funktion ... 9– 68
ERASE–Anweisung ... 9– 69
ERL–Systemvariable ... 9– 70
ERR–Systemvariable ... 9– 71
ERROR–Anweisung ... 9– 72
EXIT SUB–Anweisung ... 9– 74
EXP–Funktion ... 9– 75

FIELD–Anweisung ... 9– 76
FILES–Befehl ... 9– 78
FIX–Funktion ... 9– 79
FOR. . .NEXT–Anweisungen ... 9– 80
FRE–Funktion ... 9– 82

GET–Anweisung (Dateien) ... 9– 83
GET–Anweisung (Grafik) ... 9– 84
GOSUB– und **RETURN**–Anweisungen ... 9– 86
GOTO–Anweisung ... 9– 88

HEX$–Funktion ... 9– 89

IF. . .THEN/GOTO. . .ELSE–Anweisungen ... 9– 90
INKEY$–Funktion ... 9– 94
INPUT–Anweisung ... 9– 95
INPUT$–Funktion ... 9– 97
INPUT #–Anweisung ... 9– 98
INSTR–Funktion ... 9– 100
INT–Funktion ... 9– 102

KILL–Befehl ... 9– 103

LBOUND–Funktion ... 9– 104
LEFT$–Funktion ... 9– 105
LEN–Funktion ... 9– 106
LET–Anweisung ... 9– 107
LIBRARY–Anweisung ... 9– 108
LINE–Anweisung ... 9– 109
LINE INPUT–Anweisung ... 9– 111
LINE INPUT #–Anweisung ... 9– 113
LIST–Befehl ... 9– 114
LLIST–Befehl ... 9– 115
LOAD–Befehl ... 9– 116
LOC–Funktion ... 9– 117
LOCATE–Anweisung ... 9– 118
LOF–Funktion ... 9– 119
LOG–Funktion ... 9– 120
LPOS–Funktion ... 9– 121
LPRINT– und **LPRINT USING**–Anweisungen ... 9– 122
LSET–Anweisung ... 9– 123

MENU–Anweisung ... 9– 124
MENU–Funktion ... 9– 126
MENU ON/OFF/STOP–Anweisungen ... 9– 127
MERGE–Befehl ... 9– 128
MID$–Anweisung ... 9– 129
MID$–Funktion ... 9– 130
MKI$–, MKL$–, MKS$–, MKD$–Funktionen ... 9– 131
MOUSE–Funktion ... 9– 133
MOUSE ON/OFF/STOP–Anweisungen ... 9– 136

NAME–Befehl ... 9– 137
NEW–Befehl ... 9– 138
NEXT–Anweisung ... 9– 139

OBJECT.AX/AY–Anweisungen ... 9– 140
OBJECT.CLIP–Anweisung ... 9– 141
OBJECT.CLOSE–Anweisung ... 9– 142
OBJECT.HIT–Anweisung ... 9– 143
OBJECT.ON/OFF–Anweisungen ... 9– 145
OBJECT.PLANES–Anweisung ... 9– 146
OBJECT.PRIORITY–Anweisung ... 9– 147
OBJECT.SHAPE–Anweisung ... 9– 148
OBJECT.START/STOP–Anweisungen ... 9– 150
OBJECT.VX/VY–Anweisungen ... 9– 151
OBJECT.VX/VY–Funktionen ... 9– 152
OBJECT.X/Y–Anweisungen ... 9– 153
OBJECT.X/Y–Funktionen ... 9– 154
OCT$–Funktion ... 9– 155
ON BREAK–Anweisung ... 9– 156
ON COLLISION–Anweisung ... 9– 158
ON ERROR–Anweisung ... 9– 160
ON. . .GOSUB– und **ON. . .GOTO**–Anweisungen ... 9– 161
ON MENU–Anweisung ... 9– 163
ON MOUSE–Anweisung ... 9– 164
ON TIMER–Anweisung ... 9– 165
OPEN–Anweisung ... 9– 167
OPEN "COM1:–Anweisung ... 9– 171
OPTION BASE–Anweisung ... 9– 173

PAINT–Anweisung ... 9– 174
PALETTE–Anweisung ... 9– 175
PATTERN–Anweisung ... 9– 177
PEEK–Funktion ... 9– 178
PEEKL–Funktion ... 9– 179
PEEKW–Funktion ... 9– 180
POINT–Funktion ... 9– 181
POKE–Anweisung ... 9– 182
POKEL–Anweisung ... 9– 183
POKEW–Anweisung ... 9– 184
POS–Funktion ... 9– 185
PRESET–Anweisung ... 9– 186
PRINT–Anweisung ... 9– 188
PRINT USING–Anweisung ... 9– 191
PRINT #– und **PRINT #,USING**–Anweisungen ... 9– 196
PSET–Anweisung ... 9– 198
PTAB–Funktion ... 9– 199

PUT–Anweisung (Dateien) ... 9– 200
PUT–Anweisung (Grafik) ... 9– 201

RANDOMIZE–Anweisung ... 9– 203
READ–Anweisung ... 9– 204
REM–Anweisung ... 9– 205
RESTORE–Anweisung ... 9– 206
RESUME–Anweisung ... 9– 207
RETURN–Anweisung ... 9– 208
RIGHT$–Funktion ... 9– 209
RND–Funktion ... 9– 210
RSET–Anweisung ... 9– 212
RUN–Befehl ... 9– 213

SADD–Funktion ... 9– 215
SAVE–Befehl ... 9– 216
SAY–Anweisung ... 9– 218
SCREEN–Anweisung ... 9– 221
SCROLL–Anweisung ... 9– 223
SGN–Funktion ... 9– 224
SHARED–Anweisung ... 9– 225
SIN–Funktion ... 9– 226
SLEEP–Anweisung ... 9– 227
SOUND–Anweisung ... 9– 228
SPACE$–Funktion ... 9– 231
SPC–Funktion ... 9– 232
SQR–Funktion ... 9– 233
STICK–Funktion ... 9– 234
STOP–Anweisung ... 9– 235
STR$–Funktion ... 9– 236
STRIG–Funktion ... 9– 237
STRING$–Funktion ... 9– 238
SUB–Anweisung ... 9– 239
SWAP–Anweisung ... 9– 242
SYSTEM–Befehl ... 9– 243

TAB–Funktion ... 9– 244
TAN–Funktion ... 9– 245
TIME$–Systemvariable ... 9– 246
TIMER–Anweisung ... 9– 247
TIMER–Funktion ... 9– 248
TRANSLATE$–Funktion ... 9– 249
TRON– und **TROFF**–Befehle ... 9– 250

UBOUND–Funktion 9– 251
UCASE$–Funktion 9– 252

VAL–Funktion 9– 253
VARPTR–Funktion 9– 254

WAVE–Anweisung 9– 256
WHILE. . .WEND–Anweisungen 9– 257
WIDTH–Anweisung 9– 259
WINDOW–Anweisung 9– 262
WINDOW–Funktion 9– 265
WRITE–Anweisung 9– 267
WRITE #–Anweisung 9– 268

Anhang A: ASCII–Zeichencode–Tabellen ..... **A– 1**

Anhang B: Fehlercodes und Fehlermeldungen ..... **B– 1**

Anhang C: Reservierte Amiga Basic–Wörter ..... **C– 1**

Anhang D: Interne Zahlendarstellung ..... **D– 1**

Anhang E: Umschreibung transzendenter Funktionen ..... **E– 1**

Anhang F: Format für Maschinensprachebibliotheken ..... **F– 1**

Anhang G: Ein Beispielprogramm ..... **G– 1**

Anhang H: Sprachausgabe mit Amiga Basic ..... **H– 1**

1. Amiga Basic – Eine Übersicht

Erst wenn Sie die Möglichkeit haben, das Geschehen auf dem Bildschirm Ihres Amiga zu beeinflussen, können Sie erfolgreich mit dem Computer arbeiten. Zwar erlauben die meisten Computeranwendungen, seien es kommerzielle Programme oder Spiel–Programme, ein gewisses Maß an Ablaufsteuerung durch den Anwender. Um jedoch die gesamte Leistungsfähigkeit und Flexibilität eines Computersystems wie des Amiga ausnutzen zu können, bedarf es einer Programmiersprache.

Ihr Amiga ist eine außergewöhnliche Maschine! Damit Sie sich deren Eigenschaften nutzbar machen können, haben wir eine außergewöhnliche Programmiersprache entwickelt, das **Amiga Basic**.

Amiga Basic enthält natürlich alle Funktionen des Standardbefehlsumfanges gängiger Basic–Dialekte.

Zusätzlich erlaubt es Ihnen jedoch auch Zugriff auf die superschnelle Grafik sowie auf Tonerzeugung und Sprachsynthetisierung und viele weitere Besonderheiten des Amiga.

Warum Basic?

Da die Basic–Befehle in englischem Klartext formuliert werden, ist diese Programmiersprache leicht zu lernen und anzuwenden. Wegen der übergroßen Popularität von Basic haben die meisten Menschen, also vielleicht auch Sie, die sich schon einmal mit Computern beschäftigt haben, bereits einiges von Basic gehört.

Das bedeutet aber, daß Sie ohne große Mühe sehr schnell Ihre eigenen Programme in Amiga Basic schreiben und ausprobieren können.

Sind Sie ein Anfänger, so werden Sie natürlich zunächst die einfachen Amiga Basic–Operationen ausprobieren. Sie brauchen keine Angst vor Programmierfehlern zu haben. Amiga Basic sagt Ihnen sofort, was Sie falsch gemacht haben und Ihr Amiga nimmt Ihnen **keinen** Programmierfehler übel.

Sobald Sie über etwas Programmiererfahrung verfügen, werden Sie sicherlich all die interessanten Möglichkeiten, die der Amiga bietet, ausprobieren wollen. Dafür stellt Amiga Basic einen eindrucksvollen Satz von Programmierhilfsmitteln zur Verfügung und erlaubt zusätzlich den Zugriff auf die residenten Amiga–Bibliotheksroutinen des Betriebssystems sowie selbst geschriebene Maschinensprache–Unterprogramme.

Über dieses Handbuch

Dieses Handbuch gliedert sich in 9 Kapitel.

Kapitel 1 enthält eine kurze Einleitung, eine Übersicht über den Aufbau des Handbuches und die Besonderheiten des Amiga Basic–Interpreters.

Kapitel 2 beschreibt anhand eines Programm–Beispiels detailliert die Arbeitsweise des Amiga Basic–Interpreters.

Kapitel 3 behandelt die Grundlagen der Arbeit mit Amiga Basic, seine Betriebsarten sowie den Bildschirm.

Kapitel 4 beschreibt das Edieren und Testen von Amiga Basic–Programmen.

Kapitel 5 enthält eine Beschreibung der Datei–Ein–/Ausgabe mit Amiga Basic.

Kapitel 6 beschäftigt sich mit Unterprogrammen, Unterbrechungsverarbeitung und Speicherverwaltung.

Kapitel 7 beschreibt das Erzeugen bewegter Bilder mit dem Objekt–Editor des Amiga Basic.

Kapitel 8 behandelt die Elemente des Amiga Basic wie Zeichensatz, Sprungmarken, Konstanten, Variablen, Ausdrücke und Operatoren.

Kapitel 9 enthält das alphabetische Verzeichnis aller Befehle, Anweisungen, Funktionen und Systemvariablen sowie deren Beschreibung und stellt damit den Hauptteil dieses Handbuches dar.

Anhang A enthält die ASCII–Code–Zeichentabelle.

Anhang B enthält eine Zusammenstellung aller Fehler–Codes und –Meldungen.

Anhang C beinhaltet eine Tabelle der reservierten Worte des Amiga Basic–Interpreters.

Anhang D beschreibt die interne Darstellung von Zahlen.

Anhang E enthält Umschreibungen der transzendenten Funktionen mit vorhandenen Amiga Basic–Funktionen.

Anhang F enthält Informationen, wie eine Bibliothek von Maschinensprache–Unterprogrammen für Amiga Basic–Programme aufgebaut werden kann.

Anhang G beschreibt noch einmal detailliert das in Kapitel 2 verwendete Beispielprogramm.

Anhang H schließlich behandelt ausführlich die Grundlagen der Sprach–Synthetisierung und die Nutzung der entsprechenden Einrichtungen des Amiga zur Ausgabe künstlicher Sprache.

1.1 Spezielle Eigenschaften von Amiga Basic

Der Amiga Basic–Interpreter ist vollständig in der Assemblersprache für den Mikroprozessor MC68000 geschrieben und belegt deshalb gemessen an seiner Leistungsfähigkeit einen relativ kleinen Teil des Speichers (ca. 80 KBytes). Der Kern des Interpreters wurde über einen Zeitraum von 3 Jahren intensiv in der Praxis getestet. Amiga Basic ist insofern ein Standard–Basic, als damit die meisten in Standard–Microsoft– oder GW–Basic für andere Rechner geschriebenen Programme bearbeitet werden können.

Vereinfachung der Programmentwicklung

Wie jede andere Programmiersprache auch wird Amiga Basic ständig erweitert, verändert und verbessert, um auch neue Rechnereigenschaften zu berücksichtigen. Im folgenden sind einige der besonderen Eigenschaften des Amiga, die von Amiga Basic unterstützt werden, zusammengestellt. Eine detaillierte Beschreibung finden Sie in Kapitel 9.

Unterstützung für Amiga–Anwendersoftware

Amiga Basic stellt ihnen Hilfsmittel zur Verfügung, die Sie benötigen, um Basic–Programme zu schreiben, die so ablaufen wie professionelle, speziell für den Amiga erstellte Software. Diese Hilfsmittel sind für Software–Entwickler eine große Unterstützung, die Anwendersoftware erstellen und verkaufen wollen.

Außerdem können bekannte Programme, die in den Basic–Dialekten für Macintosh– *) oder IBM–PC *)–Rechner erstellt wurden, ohne große Probleme auf den Amiga übertragen werden.

Unterstützung der Maus

Mit der MOUSE–Funktion (s. Kapitel 9) kann Ihr Basic–Programm Maus–Betätigungen verarbeiten und darauf reagieren. Sie liefert die Bildschirm–Koordinaten des Maus–Zeigers unter den verschiedensten Bedingungen (Auswahltasten–Zustand und –Betätigung, Ziehen von Objekten).

MENU–Anweisung

Mit der MENU–Anweisung (s. Kapitel 9) können Sie die charakteristischen Amiga–Menüs erzeugen, öffnen, schließen und gewählte Menü–Punkte hervorheben, um damit Ihren Programmen ein professionelles Aussehen zu verleihen.

1.2 Leistungsfähige Sprach–Besonderheiten

Der Amiga Basic–Interpreter stellt eine Reihe von äußerst leistungsfähigen Besonderheiten zur Verfügung, die Ihren Programmen große Flexibilität verleihen. Dazu gehören:

Anweisungsblöcke

Mit Hilfe der IF–THEN–ELSE–Anweisung (s. Kapitel 9) können Sie programmierte Entscheidungen während des Programmablaufes treffen. Sie können nach der THEN–Klausel eine oder mehrere Anweisungen in einer oder auch mehreren Programmzeilen angeben und damit Ihr Programm übersichtlich strukturieren.

*) Markenzeichen der Firmen Apple bzw. IBM

Unterprogramme

Amiga Basic erlaubt Unterprogramme mit lokalen Variablen. Mit Hilfe von Unterprogrammen können Sie sich Ihre eigene Bibliothek von Basic–Programmbausteinen aufbauen, die sie bei verschiedenen Amiga Basic–Programmen einsetzen können. Dabei brauchen Sie keine Probleme wegen gleicher Variablennamen im Haupt– und Unterprogramm zu befürchten.

Globalisierung lokaler Variablen mit der SHARED–Anweisung

Umgekehrt können Sie aber auch Variablen in einem Unterprogramm als globale Variablen deklarieren, um so die Übergabe von Werten zwischen Haupt– und Unterprogramm zu erleichtern.

Ganzzahl–Arithmetik

Amiga Basic unterstützt sowohl 16– als auch 32–Bit–Ganzzahl–Arithmetik.

Gleitpunkt–Arithmetik

Gleitpunkt–Arithmetik kann mit 32 oder 64 Bit durchgeführt werden, um große Rechengenauigkeit zu erzielen.

Programme ohne Zeilennummern

Amiga Basic–Programme benötigen keine Zeilennummern mehr. Durch Zuordnen alphanumerischer oder numerischer Sprungmarken können Sie strukturierte Programme erstellen, die sich durch große Übersichtlichkeit auszeichnen.

Unterstützung von sequentiellen und Direktzugriffs–Dateien

Sie können mit Amiga Basic sowohl sequentielle als auch Direktzugriffs–Dateien auf Diskette oder Festplatte erzeugen. Erstere sind leichter zu handhaben, letztere erlauben einen wesentlich schnelleren Datenzugriff.

Geräteunabhängige Ein–/Ausgabe

Mit Hilfe der üblichen Datei–Ein–/Ausgabe–Anweisungen des Amiga Basic-Interpreters kann ein Programm die Daten–Ein–/Ausgabe an den Bildschirm, von der Tastatur, an den Drucker oder über die RS232– oder Parallel–Schnittstelle umleiten. Sie können den Drucker oder Bildschirm für Datenausgabe ebenso einfach öffnen, wie Sie eine Disk–Datei öffnen.

1.3 Unterstützung der besonderen Amiga–Möglichkeiten

Eine Reihe von Besonderheiten des Amiga Basic–Interpreters machen Gebrauch von den einzigartigen grafischen und akustischen Möglichkeiten des Amiga:

- Synchronisierte 4–Kanal–Musikwiedergabe mit den SOUND– und WAVE–Anweisungen.
- Erzeugung verständlicher Sprache mit der SAY–Anweisung und der TRANSLATE$–Funktion.
- Speichern und Wiedergewinnen von grafischen Objekten mit den GET– und PUT–Anweisungen.
- Vollständiger Satz grafischer Anweisungen: LINE, CIRCLE, PAINT, PUT, GET, AREA, AREAFILL.
- Weitgehende Unterstützung grafischer Animation (Bewegen grafischer Objekte) mit den OBJECT–Anwisungen, dem Objekt–Editor und der COLLISION–Funktion.
- Verwaltung und Aufruf von Maschinensprache–Unterprogrammen mit Hilfe der LIBRARY– und DECLARE FUNCTION–Anweisungen.
- Öffnen und Bedienen mehrerer Bildschirme und Fenster mit den SCREEN– und WINDOW–Anweisungen.
- Erstellung und Benutzung von Abroll–Menüs (pull–down menus) mit Hilfe der MENU–Anweisung.

Alle diese Anweisungen werden detailliert in Kapitel 9 beschrieben. Der Objekt–Editor wird in Kapitel 7 behandelt. Im folgenden wird für einige der genannten Anweisungen eine kurze Übersicht gegeben.

Klangerzeugung mit SOUND und WAVE

Sie können mit Amiga Basic–Programmen Klänge hoher Tonqualität für Spiele und Musikprogramme erzeugen, oder auch Anwender–Warnmeldungen ausgeben. Die SOUND–Anweisung gibt einen Ton wählbarer Frequenz, Dauer und Lautstärke aus. Zusätzlich kann noch zwischen 4 verschiedenen Tonkanälen gewählt werden. Jedem dieser Tonkanäle können Sie mit Hilfe der WAVE–Anweisung Ihre eigene, vielleicht komplexe, Wellenform zuordnen. Mit Hilfe dieser beiden Anweisungen können Sie Ihre Programme mit Musik von der Komplexität eines Streichquartetts bis zu der Simplizität eines Pfeiftones ausstatten.

Figuren zeichnen mit LINE und CIRCLE

LINE und CIRCLE sind vielseitige Anweisungen zur Erzeugung präziser grafischer Darstellungen. Mit der LINE–Anweisung können Sie eine Linie zwischen zwei Punkten mit absoluten oder relativen Koordinaten zeichnen. Mit Hilfe der B–Option für die LINE–Anweisung können Sie ein Rechteck oder Quadrat zeichen, das Sie auch noch mit einer beliebigen Farbe ausfüllen können (BF–Option).

Mit der CIRCLE–Anweisung lassen sich Ellipsen, Kreise oder Kreisbögen mit wählbarem Mittelpunkt und Radius zeichnen, optional auch in Farbe. Durch Veränderung der Aspekt–Option (Verhältnis von Bildbreite zu Bildhöhe) kann man die verschiedensten Ellipsen erzeugen.

Grafische Bildverarbeitung mit GET, PUT und SCROLL

Mit der GET–Anweisungen können Ansammlungen von Bildpunkten auf dem Bildschirm in ein Feld übertragen werden. Auf diese Weise lassen sich Grafiken schnell in den Hauptspeicher des Rechners übertragen.

Mit der PUT–Anweisung lassen sich mit GET gespeicherte Bilder wieder auf dem Bildschirm darstellen.

Mit der SCROLL–Anweisung legen Sie fest, welcher Bildschirmbereich um welchen Betrag in welche Richtung auf–, ab–, hin– oder hergerollt werden soll.

Der Objekt–Editor

Mit dem Objekt–Editor, einem in Amiga Basic geschriebenen Programm, können Sie grafische Objekte für die Animation in Spielprogrammen zeichnen. Kapitel 7 enthält Einzelheiten.

2. Mit Amiga Basic vertraut werden

Dieses Handbuch geht davon aus, daß Sie mit den Grundfunktionen des Amiga und dem Arbeiten am Arbeitstisch (Workbench) vertraut sind. Wenn nicht, verschaffen Sie sich die notwendige Übung mit Hilfe des Amiga–Benutzerhandbuches.

Um mit Amiga Basic arbeiten zu können, benötigen Sie:

- einen Amiga–Computer, betriebsbereit zusammengeschaltet,
- die Amiga Extras–Diskette zusätzlich zur Kickstart– und Workbench–Diskette.

Ehe Sie mit der Arbeit beginnen, sollten Sie sich von der Amiga Extras–Diskette zwei Sicherungskopien anfertigen.

Der Start von Amiga Basic geht in folgenden vier Schritten vor sich:

- Schalten Sie den Computer und den Monitor ein. Sobald Sie zum Einlegen der **Kickstart**–Diskette aufgefordert werden, legen Sie diese in das eingebaute Laufwerk ein.
- Nach wenigen Augenblicken werden Sie zum Einlegen der **Workbench**–Diskette aufgefordert. Legen Sie sie anstelle der Kickstart–Diskette ein und warten Sie, bis Sie das Piktogramm der Workbench–Diskette auf dem Bildschirm sehen und die Laufwerk–Kontrollampe erloschen ist.
- Legen Sie jetzt die **Amiga Extras**–Diskette in ein beliebiges Laufwerk, falls Sie über mehrere verfügen, sonst in das eingebaute Laufwerk.
- Sobald das Extras–Piktogramm auf dem Bildschirm erscheint, öffnen Sie es und öffnen Sie dann in dem Fenster das Amiga Basic–Piktogramm.

Nach wenigen Sekunden sehen Sie den Amiga Basic–Bildschirm.

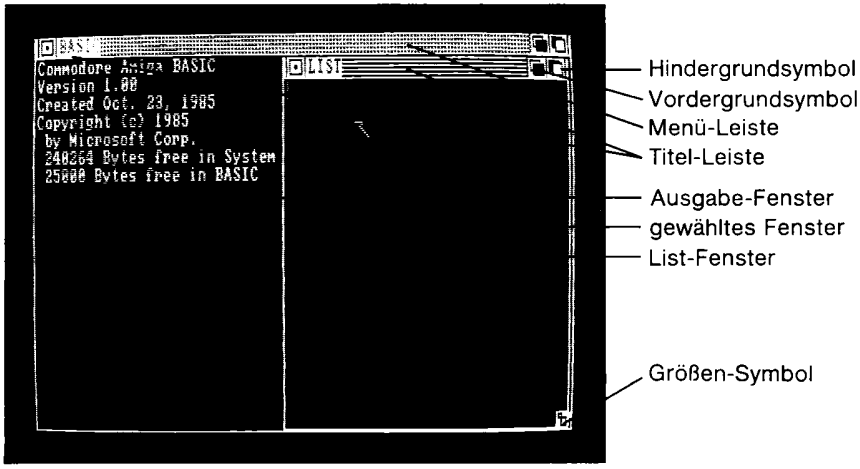
Beachten Sie In diesem Abschnitt wird davon ausgegangen, daß der Amiga Basic–Bildschirm die Originalfarben des Arbeitstisches (blauer Hintergrund, weißer Vordergrund, orange und schwarz als zusätzliche Farben) zeigt.

Zu diesem Zeitpunkt erscheint der Text–Cursor, ein vertikaler orangefarbener Strich im List–Fenster, dem rechten der beiden auf dem Amiga Basic–Bildschirm dargestellten Fenster. Der Text–Cursor gibt die Position im Fenster an, bei der das nächste auf der Tastatur eingebene Zeichen abgebildet wird. Sie können im List–Fenster entweder ein neues Programm eingeben oder ein existierendes weiter bearbeiten und ändern, wie Sie im nächsten Abschnitt noch sehen werden. Die Titelleiste im List–Fenster wird ausgeprägt dargestellt, um zu zeigen, daß dieses Fenster ausgewählt, also aktiv ist, während die Titelleiste im Ausgabe-Fenster (das linke, mit BASIC betitelte Fenster) in Geisterschrift dargestellt ist, um zu zeigen, das dieses Fenster inaktiv ist.

Das Ausgabe–Fenster von Amiga Basic zeigt Ihnen nicht nur die Ergebnisse eines Programms an, sondern erlaubt Ihnen auch die direkte Eingabe und das Ausführen von Basic–Befehlen. Sie können zu jeder Zeit, wenn Sie Befehle direkt ausführen lassen wollen, die Titelleiste des Ausgabe–Fensters wählen. Amiga Basic quittiert dies mit der **Ok**–Anzeige und stellt den Text–Cursor bereit.

Um die Menü–Titel in der Menü–Leiste darzustellen, wählen Sie das Ausgabe-Fenster und halten dann die Menü–Taste der Maus (die rechte Taste) gedrückt.

So sieht also der Amiga Basic–Bildschirm aus:



2.1 Ein praktisches Beispiel

Sie können jetzt durch die Bearbeitung eines praktischen Beispiels mit dem Amiga Basic–Interpreter vertraut werden.

Um den Inhalt der Amiga Extras–Diskette anzuzeigen,

- wählen Sie das Ausgabe–Fenster.

Wenn die Ok–Bereitschaftsanzeige erscheint,

- geben Sie ein:

```
files
```

und

- drücken Sie die RETURN–Taste.

Sie sehen jetzt im Ausgabe–Fenster eine Liste mit Datei– und Verzeichnisnamen. Das Fenster füllt sich und der Inhalt wird dann zeilenweise nach oben gerollt, je mehr Informationen angezeigt werden. Um das Aufrollen anzuhalten, drücken Sie die **rechte Amiga–Taste** zusammen mit der **S**–Taste. Zur Fortsetzung drücken Sie jede beliebige Taste.

Um die Dateien in einem der Verzeichnisse anzuzeigen, geben Sie den Befehl **files** gefolgt von dem gewünschten Verzeichnisnamen, in Anführungsstriche (") eingekleidet, ein. Befindet sich die betreffende Diskette in einem externen Laufwerk, so muß vor dem Verzeichnisnamen noch die Laufwerksbezeichnung angegeben werden. Wenn sich z.B. die **Extras**–Diskette im Laufwerk 1 befindet, listet der folgende Befehl alle Dateien des Verzeichnisses **BasicDemos**:

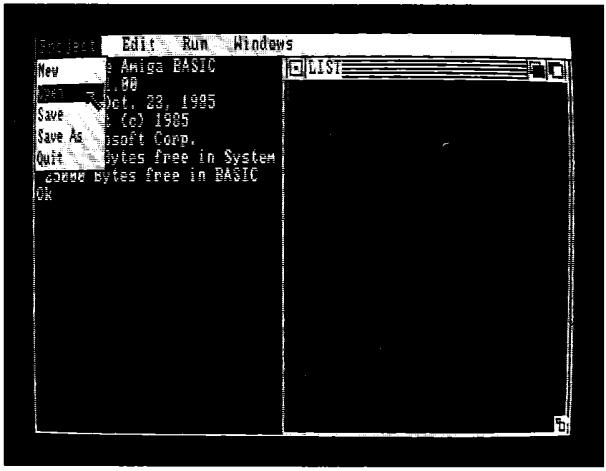
```
files "df1:basicdemos"
```

Das Programm Picture laden

Beginnen Sie jetzt, indem Sie das Programm Picture laden, das in der BasicDemos–Ablage (oder –Verzeichnis) gespeichert ist. Picture ist ein in Amiga Basic geschriebenes kleines Demonstrationsprogramm:

- Drücken Sie die Menütaste der Maus und zeigen Sie auf den Menü–Titel Project in der Menü–Leiste. Es werden die Menü–Punkte New, Open, Close, Save, Save As und Quit angezeigt.

- Wählen Sie **Open**.



Im Ausgabe–Fenster wird ein Kommunikationsfenster eingeblendet.

- Wählen Sie das Symbolfeld "Name of program to load" (Name des zu ladenden Programms)

- Geben Sie ein:

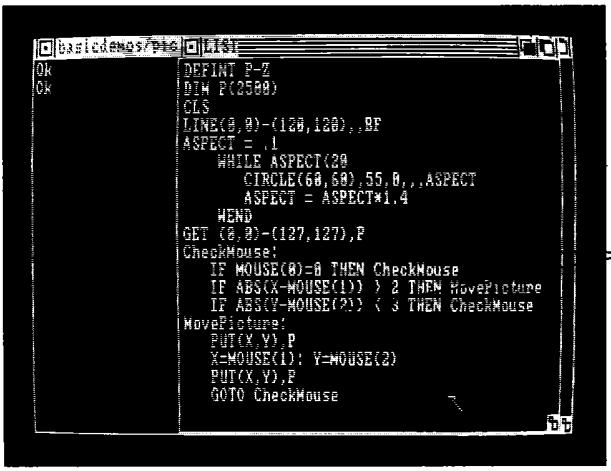
```
BasicDemos/Picture
```

- Wählen Sie das Symbolfeld OK oder drücken Sie die RETURN–Taste.

Nähere Informationen zur Schreibweise von Datei– und Verzeichnisnamen finden Sie im Kapitel 5.2 sowie im AmigaDOS–Benutzerhandbuch.

Die Programmliste des Programms Picture Sobald das Programm geladen ist, wird es im List–Fenster aufgelistet. Der Name des Ausgabe–Fensters ändert sich von BASIC in Basic Demos/Picture.

Vielleicht haben Sie erwartet, daß jede Programmzeile mit einer Zeilennummer beginnt. Amiga Basic benötigt keine Zeilennummern, akzeptiert sie aber, wenn sie vorhanden sind. Um auf eine bestimmte Zeile Bezug zu nehmen, brauchen Sie dieser Zeile nur eine Sprungmarke oder eine Nummer voranzustellen. Das Picture–Programm hat, wie Sie sehen können, keine Zeilennummern, dafür aber zwei Sprungmarken: **CheckMouse** und **MovePicture**:



Sprungmarken und Zeilennummern stellen Einsprung–Punkte für GOTO– oder GOSUB–Anweisungen in anderen Programmteilen dar.

- Wählen Sie jetzt das Ausgabe–Fenster und geben Sie ein:

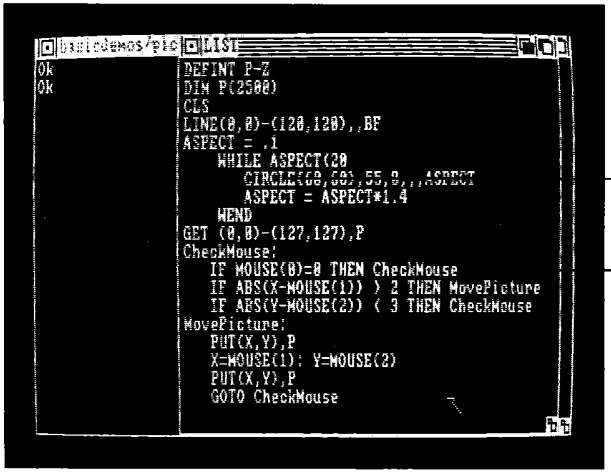
```
list CheckMouse
```

- Drücken Sie die RETURN–Taste.

Sie sehen, daß der Inhalt des List–Fensters bis zu der Sprungmarke **CheckMouse:** aufgerollt wird. Wenn Sie jetzt im List–Fenster edieren wollen, müssen Sie es zuerst wählen.

Reservierte Worte in Großschrift Amiga Basic–Programme sind auf dem Bildschirm leicht zu lesen, weil die reservierten Wörter, also die Basic–Schlüs-

selwörter, automatisch in Großschrift umgewandelt werden. Andere Teile wie Variablen– und Sprungmarken–Namen bleiben so, wie sie eingegeben wurden:



Reservierte Amiga-Basic-Wörter werden in Großschrift dargestellt

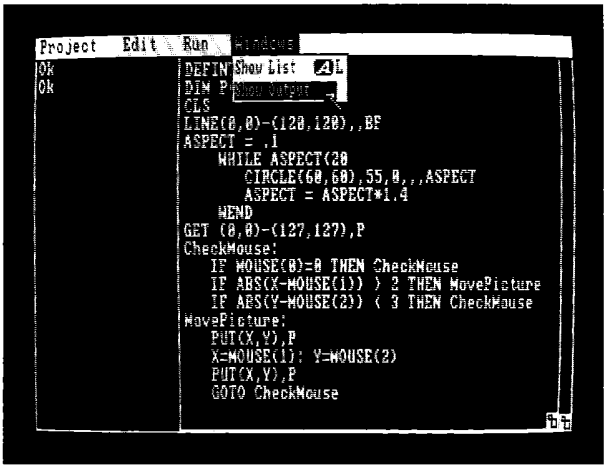
Andere Wörter werden wie angegeben dargstellt.

Beachten Sie, daß die reservierten Wörter erst in Großschrift angezeigt werden, wenn Sie die Zeilen anzeigen, nicht schon bei der Eingabe.

Was das Programm Picture macht

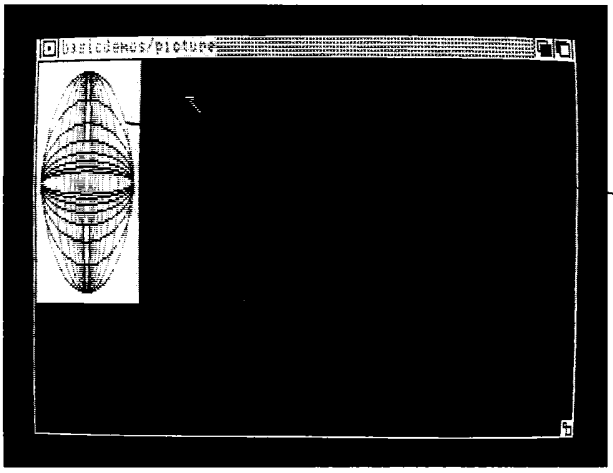
Starten Sie jetzt das Programm wie folgt:

- Um das Ausgabe–Fenster über dem List–Fenster zu öffnen, wählen Sie **Show Output Window** (zeige Ausgabe-Fenster) aus dem **Windows**-Menü.



- Wählen Sie **Start** aus dem **Run**–Menü.

Im Ausgabe–Fenster erscheint jetzt das Programm–Ergebnis: die strukturierte Darstellung eines Ellipsoids. Dieses Bild können Sie über den Bildschirm bewegen, indem Sie die Auswahltaste der Maus irgendwo im Ausgabe–Fenster drücken. Versuchen Sie es einmal.



Die Ausgabe von Picture

Das Programm anhalten

Das Programm Picture läuft solange, bis Sie es anhalten.

- Wählen Sie **Stop** aus dem **Run**–Menü.

- Wählen Sie **Show List Window** (List-Fenster zeigen) aus dem **Windows**-Menü. Damit wird das List–Fenster in den Vordergrund gebracht. Um das Programm im List–Fenster edieren zu können, müssen Sie dieses Fenster zuerst wählen.

Cursor–Bewegungen im List–Fenster

Um den Text–Cursor im List–Fenster von einer Zeile zur anderen zu bewegen, drücken Sie innerhalb dieses Fensters die Auswahltaste der Maus und drücken Sie dann eine der Vertikal–Pfeiltasten rechts unten auf der Tastatur.

Um den Cursor innerhalb einer Zeile nach rechts oder links zu bewegen, drücken Sie eine der Horizontal–Pfeiltasten rechts unten auf der Tastatur.

Um das ganze Programm fensterweise zu durchblättern, drücken Sie eine der Vertikal–Pfeiltasten zusammen mit der SHIFT–Taste.

Beachten Sie: Wann immer Sie im weiteren Verlauf dieses Handbuches auf die Bezeichnung zweier Tasten, verbunden mit einem Bindestrich stoßen, wie z.B. SHIFT–Pfeil nach oben, so beutet dies, daß Sie die erste Taste niederhalten und dann die zweite Taste drücken, im Beispiel also die SHIFT–Taste niederhalten und dann die Pfeil–nach–oben–Taste drücken.

Um also vorwärts fensterweise durch das Programm zu blättern, drücken Sie SHIFT–Pfeil nach unten, um rückwärts zu blättern, SHIFT–Pfeil nach oben.

Um direkt in die erste Programmzeile zu gelangen, drücken Sie ALT–Pfeil nach oben und um direkt zur letzten Programmzeile zu kommen, ALT–Pfeil nach unten.

ALT–Pfeil nach rechts setzt den Cursor an das Ende der aktuellen Programmzeile, ALT–Pfeil nach links setzt ihn an den Zeilenanfang.

SHIFT–Pfeil nach rechts plaziert den Cursor an den Anfang des letzten Viertels einer Programmzeile, SHIFT–Pfeil nach links an das Ende des ersten Viertels. Dies ist dann hilfreich, wenn sehr lange Programmzeilen bearbeitet werden sollen.

Anhang G enthält eine zeilenweise Beschreibung des Beispielprogramms Picture.

Ein Basic–Programm edieren

Das Edieren eines Amiga Basic–Programms entspricht der Textbearbeitung mit einem Textprogramm. Die Eingabe von Text im List–Fenster erfolgt mit den Funktionen **Cut** (abschneiden), **Copy** (kopieren) und **Paste** (ersetzen) aus dem **Edit**–Menü.

Um neuen Text einzufügen, plazieren Sie zunächst den Cursor an die Einfügestelle, indem Sie den Zeiger dorthin bewegen und dann die Auswahltaste drücken. Anschließend geben Sie Ihren Text ein.

Mit der BACKSPACE–Taste können Sie Zeichen links vom Cursor löschen. Um Zeichen rechts vom Cursor zu löschen, drücken Sie die DEL–Taste.

Um ein Wort zu wählen, zeigen Sie mit dem Zeiger d'rauf und drücken die Auswahltaste der Maus zweimal kurz.

Zur Auswahl eines ganzen Textblockes drücken Sie am Anfang die Auswahltaste, bewegen dann den Zeiger zum Ende des Blocks und halten hier zuerst die SHIFT–Taste nieder und drücken dann die Auswahltaste der Maus erneut. Einen so ausgewählten Block können Sie jetzt mit **Cut** löschen oder mit **Copy** an eine andere Stelle im Programm kopieren.

Um das List–Fenster zu verbreitern, damit Sie die volle Zeilenlänge sehen könen, gehen Sie folgendermaßen vor:

- Zeigen Sie auf die Titelleiste und halten Sie hier die Auswahltaste der Maus gedrückt. Ziehen Sie jetzt das gesamte List–Fenster an den linken Bildschirmrand.

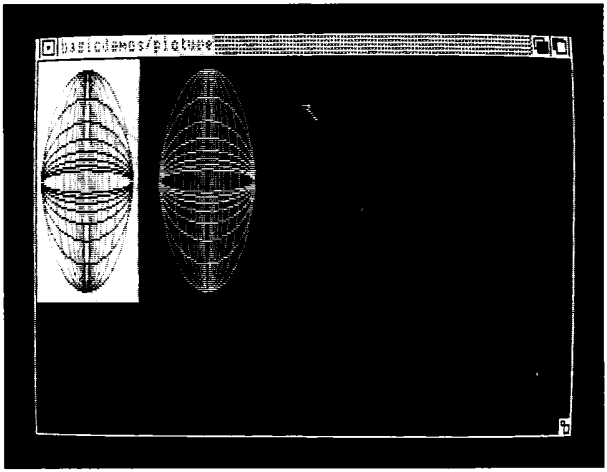
- Geben Sie die Auswahltaste wieder frei und zeigen mit dem Zeiger auf das Größen–Symbol in der rechten unteren Fensterecke. Halten Sie hier wieder die Auswahltaste gedrückt und ziehen Sie das Fenster so weit auseinander, bis Sie die volle Programmzeilenlänge sehen können.

- Lassen Sie die Auswahltaste wieder los, wenn das Fenster die gewünschte Größe hat.

Edieren des Beispielprogramms Picture

Sie haben jetzt die Gelegenheit, am Beispielprogramm Picture erste Edier-Erfahrungen zu machen und dabei einiges über die graphischen Anweisungen von Amiga Basic zu lernen. Sie brauchen keine Angst zu haben, das Programm unbrauchbar zu machen oder zu verlieren. Auf der Diskette befindet sich eine Kopie unter dem Namen Picture2.

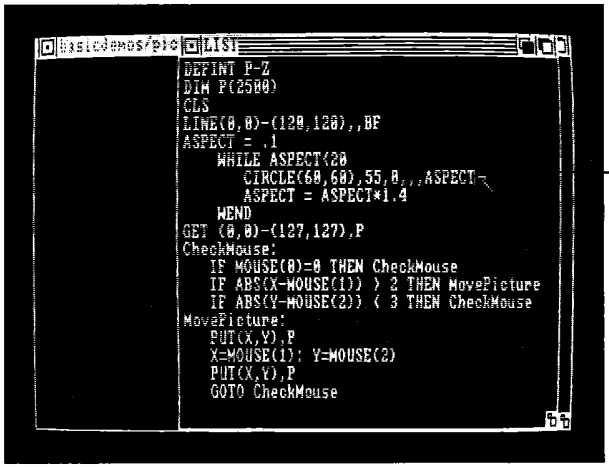
Wenn Sie experimentierfreudig sind, machen Sie ruhig Ihre eigenen Programmänderungen. Versuchen Sie mit den unten beschriebenen Schritten das folgende Programmergebnis zu erzielen:



Eine Programmzeile hinzufügen Beginnen Sie mit dem Hinzufügen einer Programmzeile, die ein zweites Bild erzeugt:

- Suchen Sie im List–Fenster, bis Sie folgende Programmzeile gefunden haben:

```
CIRCLE(60,60),55,,,,ASPECT
```



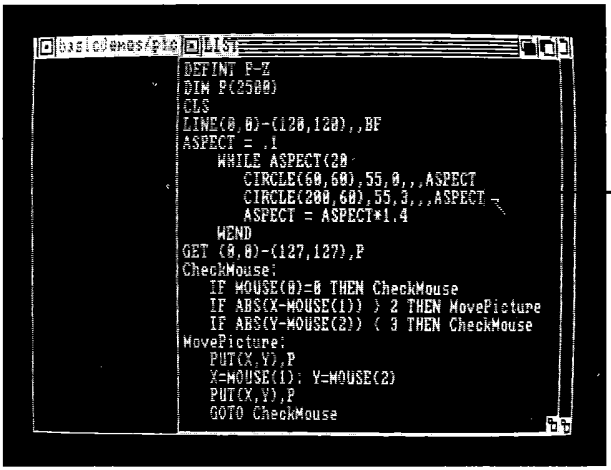
Suchen Sie die Programmzeile, die die erste Ellipse zeichnet

- Drücken Sie am Zeilenende die Auswahltaste, um den Cursor dorthin zu positionieren
- Drücken Sie die RETURN–Taste, um eine neue Zeile einzurichten.

Jetzt können Sie die neue Zeile eingeben. Beachten Sie, daß Amiga Basic den Cursor nach der unmittelbar darüberstehenden Anweisung ausrichtet, um Ihnen die Eingabe führender Leerstellen zu ersparen.

- Geben Sie jetzt ein:

```
CIRCLE(200,60),55,3,,,ASPECT
```



Geben Sie diese Programmzeile ein, um die zweite Ellipse

Diese Anweisung zeichnet eine Ellipse um den Mittelpunkt bei der Position 200,60 (200 Bildpunkte vom linken Bildschirmrand, 60 Bildpunkte vom oberen Bildschirmrand) mit dem Radius von 55 (Bildpunkten) und einem Bildverhältnis, das mit der Variablen ASPECT angegeben wird. 3 ergibt die Farbe orange, 0 die Farbe blau, vorausgesetzt, Sie arbeiten mit den Original–Arbeitstischfarben. Bei jeder Ausführung der WHILE–Programmschleife wird eine neue Ellipse mit anderem Bildverhältnis (ASPECT) gezeichnet. Die einzelnen Ellipsen bilden schließich das Abbild des Sphäroiden.

- Wählen Sie jetzt **Start** aus dem **Run**–Menü, um das Programm zu starten.

Korrektur von Fehlern

Während der Programm–Edierung kann es durchaus zu Eingabefehlern kommen. Findet Amiga Basic während eines Programmlaufes einen Fehler, so wird das Programm abgebrochen und die Fehlerbeschreibung wird in einem Kommunikationsfenster angezeigt. Amiga Basic stellt dann außerdem sicher, daß das List–Fenster sichtbar wird und rollt die Programmliste ab, damit die

fehlerhafte Zeile sichtbar wird. Die fehlerverursachende Anweisung wird mit einem orangefarbenen Rechteck eingerahmt. Sie können jetzt zur Fehlerbeseitigung die Zeile edieren und das Programm neu starten. Dieses Verfahren wird "Debugging" genannt.

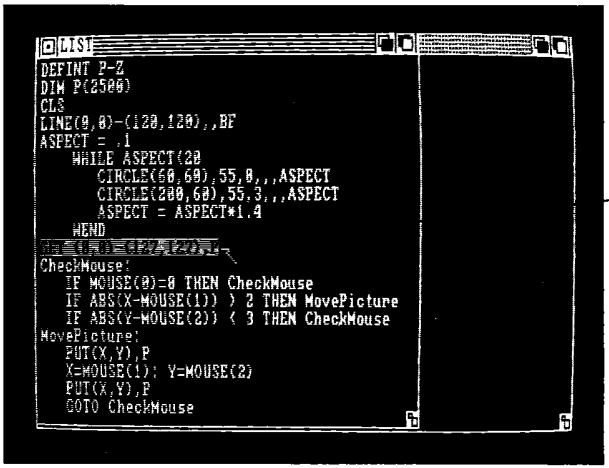
Ersetzen einer Programmzeile

Trotz der Programmänderung können Sie nur das linke Bild mit der Maus bewegen. Nun soll das Programm so gändert werden, daß beide Bilder mit der Maus gezogen werden können:

- Wenn das Programm noch läuft, wählen Sie **Stop** aus dem **Run**-Menü, um es anzuhalten.

Wählen Sie jetzt **Show List**. Die Position des List–Fensters ändert sich jedoch nicht.

- Zeigen Sie auf den äußerst linken Rand der GET–Anweisung, drücken Sie die Auswahltaste der Maus und ziehen Sie den Zeiger bis zum Zeilenende, dadurch wird die gesamte Zeile in orange hervorgehoben. Sie ist damit ausgewählt.



Wählen Sie die GET-Anweisung

- Wählen Sie jetzt **Cut** aus dem **Edit**–Menü, um die ausgewählte Zeile zu löschen.

- In der nun leeren Zeile geben Sie folgendes ein:

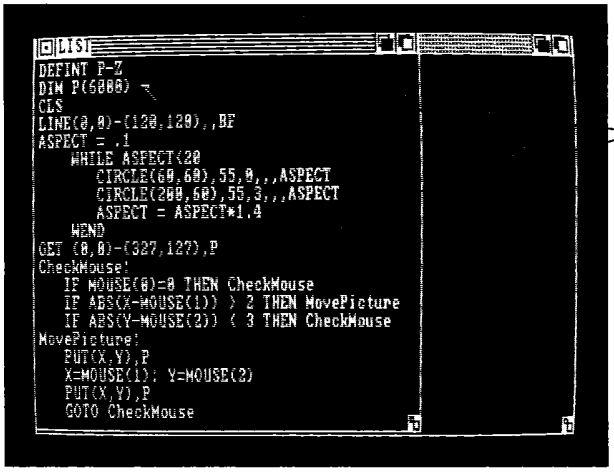
```
GET(0,0)-(327,127),P
```

Mit dieser neuen GET–Anweisung vergößern Sie den mit der Maus bewegbaren Bildausschnitt.

Ändern Sie jetzt außerdem die DIM–Anweisung, um ein Feld mit 6000 anstatt 2500 Elementen zu dimensionieren:

- Positionieren Sie den Cursor auf die DIM–Anweisung.
- Wählen Sie den Teil der Anweisung, der auf 2500 lautet, aus, und löschen Sie ihn mit **Cut** aus dem **Edit**–Menü
- Geben Sie in den freien Raum zwischen den Klammern 6000 ein, so daß die Zeile jetzt lautet:

```
DIM P(6000)
```

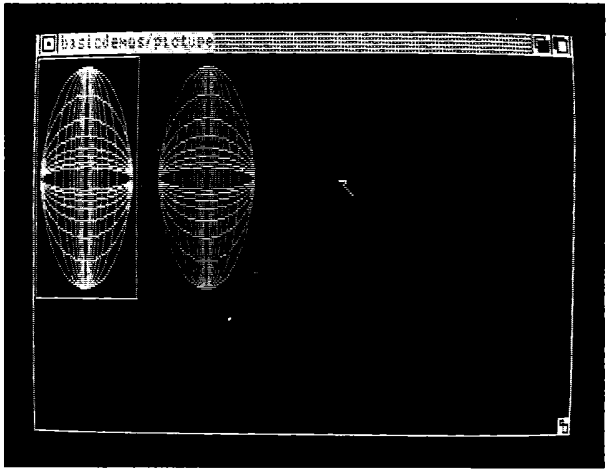


- Wählen Sie **Start** aus dem **Run**–Menü, um das Programm zu starten.

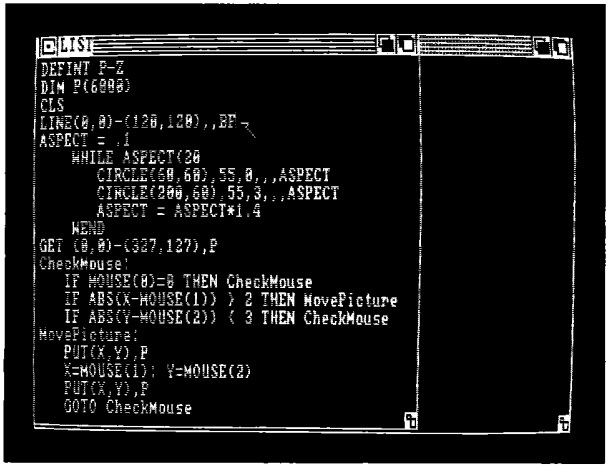
Wenn Sie jetzt auf eines der Bilder zeigen, die Auswahltaste der Maus drücken und diese bewegen, bewegen sich beide Bilder mit.

Inverse Bilddarstellung

Jetzt soll das erste Bild in weiß auf blauem Hintergrund dargestellt werden, also so:



- Wenn das Programm noch läuft, halten Sie es mit **Stop** an und öffnen Sie das List–Fenster.
- Suchen Sie die Programmzeile mit der LINE–Anweisung.
- Positionieren Sie den Cursor direkt hinter die Zeichen BF an das Ende der Zeile, indem Sie dorthin zeigen und die Auswahltaste der Maus drücken.



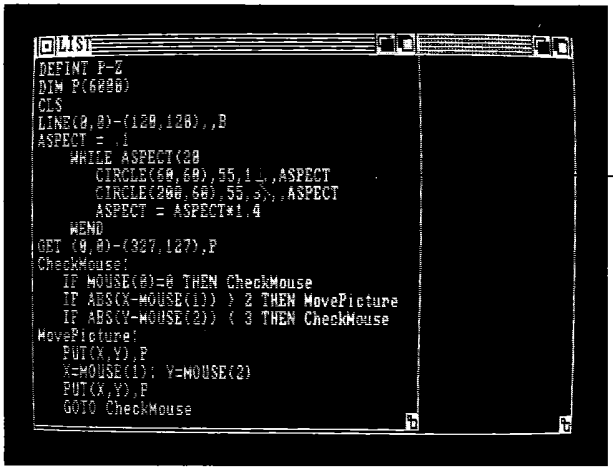
- Drücken Sie die BACKSPACE–Taste einmal, um das Zeichen F zu löschen.

Damit wird das Innere des Rechteckes, innerhalb dem das Bild dargestellt wird, in blau und nicht in weiß dargestellt.

- Suchen Sie die Zeile

```
CIRCLE(60,60),55,0,,,ASPECT
```

- Positionieren Sie den Cursor hinter die Ziffer 0.
- Drücken Sie einmal die BACKSPACE–Taste, um die 0 zu löschen.
- Geben Sie 1 ein als neuen Farbcode.



Fügen Sie eine 1 ein

Damit werden die Ellipsen in weiß anstatt in blau gezeichnet.

- Starten Sie mit **Start** das Programm neu, um das Ergebnis zu sehen.

Damit sind alle Programmänderungen vollständig.

Einzelschritt–Abarbeitung des Programms

Damit Sie besser mit dem Beispielprogramm Picture vertraut werden, soll jetzt eine häufig angewandte Technik zur Fehlersuche beschrieben werden: Die Einzelschritt–Abarbeitung eines Programms.

- Wenn das Programm noch läuft, halten Sie es mit **Stop** an.

- Wählen Sie das Ausgabefenster, indem sie die Auswahltaste der Maus an beliebiger Stelle in diesem Fenster drücken. Amiga Basic quittiert mit der Ok–Bereitschaftsanzeige.

- Geben Sie ein

```
end
```

und drücken Sie die RETURN–Taste.

- Wählen Sie **Step** aus dem **Run**–Menü. Damit wird die erste Programmzeile ausgeführt und dann das Programm angehalten.

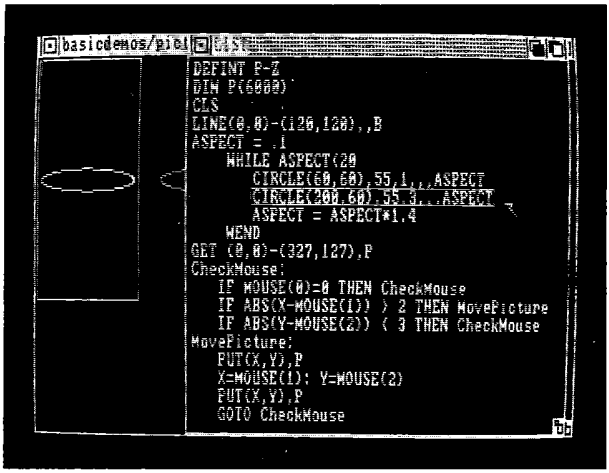
- Wählen Sie **Show List** aus dem **Windows**–Menü, um das List-Fenster auf der rechten Bildschirmseite zu öffnen und zu aktivieren.

Jede Anweisung wird nach ihrer Ausführung jetzt im List–Fenster angezeigt. Das Ausgabe–Fenster wird gewählt, damit aller eingegebener Text hier sichtbar wird.

- Wählen Sie jetzt **Step** erneut oder drücken Sie die rechte Amiga–Taste zusammen mit der T–Taste.

Die nächste Zeile wird ausgeführt und angezeigt und das Programm wird erneut angehalten. Da bis jetzt keine Ausgabe erfolgt, können Sie auch noch nichts sehen.

Wählen Sie jetzt weiter **Step** und verfolgen Sie die schrittweise Abarbeitung des Programms. Wenn der Teil, bei dem die Ellipse gezeichnet wird, erreicht wird, können Sie sehen, wie das Bild erstellt wird. Bei jedem Durchlauf durch die WHILE–Schleife wird eine neue Ellipse mit anderem Bildverhältnis gezeichnet.



- Geben Sie, nachdem die ersten Ellipsen gezeichnet sind, folgendes im Ausgabe–Fenster ein:

  print aspect

- Drücken Sie die RETURN–Taste.

Der aktuelle Wert der Variablen für das Bildverhältnis, mit dem die Ellipsen gezeichnet werden, wird im Ausgabefenster angezeigt.

Obwohl ja eigentlich das Beispielprogramm keine Fehler enthält, die mit dieser Methode lokalisiert werden könnten, zeigt sie jedoch das Verfahren anschaulich. Die Möglichkeit, zu jedem Programmausführungszeitpunkt Basic–Befehle im Ausgabe–Fenster eingeben zu können, wird als "Befehlseingabe im Direktmodus" bezeichnet. Amiga Basic führt solche Befehle sofort aus und zeigt das Ergebnis, falls eines existiert, auch sofort an. Weitere Informationen zum Direktmodus finden Sie in Kapitel 3.2.

- Fahren Sie jetzt mit der schrittweisen Abarbeitung des Beispielprogramms fort. Lassen Sie sich zwischendurch die Inhalte weiterer Variablen anzeigen.

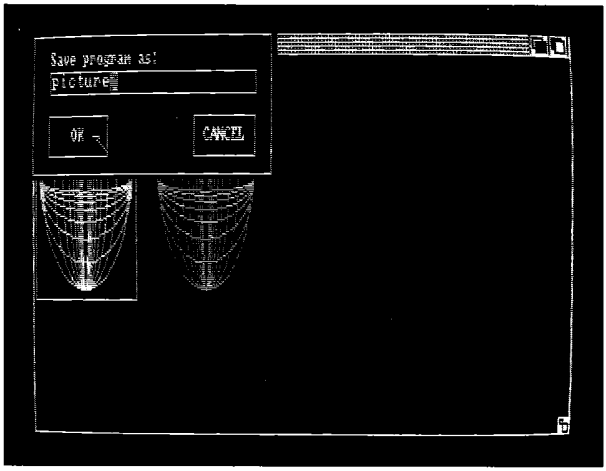
- Wenn Sie die schrittweise Abarbeitung abbrechen wollen und den Rest des Programms lieber vollständig ablaufen lassen wollen, wählen Sie **Continue** (fortsetzen) aus dem **Run**–Menü.

Das Programm auf Diskette speichern

Wenn Sie ein eingegebenes oder geändertes Programm auf Diskette speichern und gleichzeitig die alte Version erhalten wollen, wählen Sie **Save As** (speichern als) aus dem **Project**–Menü.

Ist das Programm erst einmal auf Diskette abgelegt, so können Sie es zu jedem beliebigen Zeitpunkt wieder laden und ausführen.

- Wählen Sie **Save As** aus dem **Project**–Menü. Es erscheint ein Kommunikationsfenster:



Amiga Basic nimmt an, daß Sie das Programm unter seinem gegenwärtigen Namen (Picture) und in derselben Form (üblicherweise das Binärformat), wie Sie es geladen haben, speichern wollen.

Sie können den Namen ändern, wenn Sie wollen. Der einfachste Weg ist jedoch, das **OK**–Symbolfeld zu wählen.

Damit haben Sie jetzt zwei Versionen des Programms Picture auf der Diskette. Die unveränderte Version unter dem Namen Picture2 und das veränderte unter dem Namen Picture. Ebensogut hätten Sie aber auch den Namen in Meinprogramm oder jeden anderen gültigen Namen ändern können. Damit würde auch das Programm Picture in seiner ursprünglichen Form auf Diskette erhalten bleiben.

2.2 Beenden von Amiga Basic und Rückkehr zum Arbeitstisch (Workbench)

- Wählen Sie **Quit** (aufhören) aus dem **Project–Menü**.

Damit haben Sie Ihre erste praktische Arbeit mit Amiga Basic beendet.

Sie befinden sich jetzt wieder auf dem Arbeitstisch (Workbench) und können mit dem Amiga eine andere Arbeit beginnen. Dennoch haben Sie in kurzer Zeit eine Menge über die Programmiersprache Amiga Basic gelernt. Sie haben gelernt,

- ein vorhandenes Programm von Diskette zu laden,
- ein Programm im List–Fenster zu edieren,
- mit einigen Basic–Anweisungen und –Funktionen zu arbeiten,
- ein Amiga Basic–Programm zu speichern.

Im Kapitel 3 finden Sie die Grundlagen der Arbeit mit Amiga Basic. Hier wird auch in einem separaten Abschnitt der Amiga Basic–Bildschirm detailliert beschrieben. Sie werden beim Durcharbeiten des Kapitels 3 viele Fakten, die Sie beim Bearbeiten des Beispielprogramms gelernt haben, wiederfinden, werden aber natürlich auch viel Neues dazulernen. Wie bei allen Amiga–Programmen können Sie weder dem Computer noch dem Amiga Basic–Interpreter durch Dateneingabe, Mausbedienung oder Fehlerverursachung Schaden zufügen. Deshalb experimentieren Sie ruhig mit Amiga Basic und probieren Sie alle Funktionen und Eigenschaften des Bildschirms aus.

Kurze Zusammenfassung der Programmdatei–Befehle

Im folgenden werden für alle Aufgaben, die Sie während dieser ersten praktischen Arbeit mit Amiga Basic erledigt haben, die Basic–Befehle angegeben, die Sie alternativ zum Aufruf von Menü–Punkten mit der Maus im Direktmodus verwenden können:

Ein existierendes Programm laden

Zum Laden geben Sie folgenden Befehl ein:

```
load "Dateiname"
```

Das geladene Programm edieren

Zum Edieren eines geladenen Programms oder zum Eingeben eines neuen Programms geben Sie ein:

```
list [Sprungmarke]
```

`list` ruft den Bildschirmeditor von Amiga Basic auf. Wenn Sie eine Sprungmarke angeben, so wird diese Zeile ganz oben im List–Fenster angezeigt und alle ggf. folgenden Zeilen, bis das Fenster gefüllt ist.

Ein geladenes Programm ausführen

Zum Starten geben Sie ein:

```
run
```

Um das Programm anzuhalten, drücken Sie CTRL–C. Um sich z.B. Feldvariableninhalte anzuschauen, können Sie

```
for i=1 to 20:print a(i):next
```

eingeben.

Zum Fortsetzen eines angehaltenen Programms können Sie

```
cont
```

eingeben.

Amiga Basic beenden

Um die Arbeit mit Amiga Basic zu beenden und zum Arbeitstisch (Workbench) zurückzukehren, geben Sie ein:

```
system
```

Ein Programm auf Diskette speichern

Wenn das im Hauptspeicher befindliche Programm geändert wurde und Sie Amiga Basic verlassen wollen, ohne das Programm zu speichern, erhalten Sie die Anzeige

```
Current program is not saved
Do you want to save it before proceeding?
```

(Aktuelles Programm nicht gespeichert
Wollen Sie es speichern, ehe Sie fortfahren?)

Wenn Sie hier das **YES**–Feld wählen, erhalten Sie in einem Kommunikationsfenster die Aufforderung zur Eingabe des Dateinamens, worauf das Programm gespeichert wird. Ansonsten wird Amiga Basic ohne Speicherung beendet.

3. Arbeiten mit Amiga Basic

Dieses Kapitel enthält die grundlegenden Informationen, die Sie für ein erfolgreiches Arbeiten mit dem Amiga Basic–Interpreter benötigen. Es wird beschrieben, wie Amiga Basic aufgerufen und beendet wird, wie Programmdateien geladen und gespeichert werden und wie die verschiedenen Betriebsmodi von Amiga Basic eingesetzt werden können. In einem weiteren Abschnitt wird der Amiga Basic-Bildschirm beschrieben.

3.1 Grundlegende Funktionen

In diesem Abschnitt wird der Aufruf und das Beenden des Amiga Basic–Interpreters sowie das Laden und Speichern von Amiga Basic–Programmen beschrieben.

Aufruf von Amiga Basic

Sie können den Amiga Basic–Interpreter auf zweierlei Weise aufrufen:

1. Öffnen Sie das Piktogramm für CLI im Fenster der **System**–Schublade auf dem Arbeitstisch. Das Fenster des AmigaDOS–Befehlsinterpreters wird geöffnet und Sie geben über Tastatur

   ```
   AmigaBASIC
   ```

   ein. Drücken Sie zum Schluß die RETURN–Taste.
   Sie können aber auch gleich den Namen des abzuarbeitenden Basic-Programms mit angeben:

   AmigaBASIC MeinProg oder:
   run [Pfad]**AmigaBASIC** [Pfad]**Meinprog**

2. Zeigen Sie auf das Amiga Basic–Piktogramm im Fenster der Extras–Diskette und drücken Sie die Auswahltaste der Maus kurz zweimal. Dadurch wird zusätzlich das voreingestellte Amiga Basic–Programm geladen und gestartet.

Beenden von Amiga Basic und Rückkehr zum Arbeitstisch

Die Arbeit mit Amiga Basic kann auf zweierlei Weise beendet werden:

1. Wählen Sie **Quit** aus dem **Projekt**–Menü.

2. Geben Sie im gewählten Ausgabe–Fenster ein:

   ```
   system
   ```

   und drücken Sie die RETURN–Taste oder verwenden Sie den SYSTEM-Befehl in einem Amiga Basic–Programm.

Ein Programm laden

Um ein Programm ablaufen lassen zu können, muß es im Hauptspeicher sein. Es gibt verschiedene Möglichkeiten, ein Amiga Basic–Programm in den Hauptspeicher zu laden.

- Zeigen Sie auf dem Arbeitstisch auf das Piktogramm für das gewünschte Amiga Basic–Programm und drücken Sie die Auswahltaste der Maus kurz zweimal hintereinander. Damit wird sowohl der Amiga Basic–Interpreter als auch das auszuführende Programm geladen und gestartet.

- Wenn der Amiga Basic–Interpreter bereits aktiviert wurde, wählen Sie **Open** aus dem **Project**–Menü. Damit werden Sie in einem Kommunikationsfenster aufgefordert, den Namen des zu ladenden Programms einzugeben. Wählen Sie hier das Titel–Symbolfeld und geben Sie dann den Namen ein. Drücken Sie zum Abschluß entweder die RETURN–Taste oder wählen Sie das OK–Symbolfeld.

- Geben Sie im gewählten Ausgabe–Fenster die Befehle

  ```
  load"Dateiname"
  ```

  oder

  ```
  run"Dateiname"
  ```

  ein, wobei für *Dateiname* der Name der Programmdatei einzusetzen ist.

- Von einem laufenden Amiga Basic–Programm aus kann mit Hilfe der CHAIN–Anweisung (s. dort in Kapitel 9) ein weiteres Programm geladen und gestartet werden.

Ein Programm speichern

Um ein neues Programm zu speichern, können Sie entweder **Save As** aus dem **Project**–Menü wählen oder den Befehl SAVE im Ausgabe–Fenster eingeben. Ein bereits schon einmal früher gespeichertes und jetzt verändertes Programm können Sie entweder mit **Save** aus dem **Project**–Menü oder durch Eingabe des SAVE–Befehls im Ausgabe–Fenster speichern.

Amiga Basic speichert alle Programme in komprimierter Form, es sei denn, Sie geben beim SAVE–Befehl explizit etwas anderes an (s. dort in Kapitel 9).

3.2 Betriebsarten von Amiga Basic

Wenn Sie den Amiga Basic–Interpreter aufrufen, wird das Ausgabe–Fenster mit dem Titel **BASIC** geöffnet. Es können jetzt in diesem Fenster Befehle eingegeben werden. Dazu stehen drei Betriebsmodi des Interpreters zur Verfügung:

- Direktmodus
- Ediermodus
- Programmodus

Das List–Fenster wird aktiviert, sobald Amiga Basic beginnt zu arbeiten.

Direktmodus

Beim Direktmodus werden die Amiga Basic–Befehle oder –Anweisungen nicht im Speicher abgelegt, sondern direkt nach der Eingabe im AusgabeFenster ausgeführt. Ergebnisse arithmetischer oder logischer Operationen werden direkt angezeigt und außerdem zur späteren Verwendung gespeichert. Die Anweisung selbst geht jedoch nach ihrer Ausführung verloren. Der

Direktmodus ist bei der Fehlersuche und zur Verwendung des Amiga Basic-Interpreters als Rechner für schnelle Berechnungen, die kein vollständiges Programm erfordern, hilfreich. Um den Direktmodus einzuschalten, müssen Sie auf eine beliebige Stelle im Ausgabe–Fenster zeigen und die Auswahltaste der Maus drücken.

Ediermodus

Im Ediermodus sind Sie, sobald Sie im List–Fenster arbeiten. Die eingegebenen Anweisungen und Befehle werden solange nicht ausgeführt, wie kein RUN–Befehl gegeben wird, oder nicht **Start** aus dem **Run**–Menü gewählt wird.

Programmodus

Sobald ein Programm gestartet ist, befindet sich Amiga Basic im Programmodus. Während der Programmausführung können Sie weder Direktmodus–Befehle eingeben, noch können Sie neue Programmzeilen im List–Fenster eingeben.

3.3 Der Amiga–Basic–Bildschirm

Der Amiga Basic–Bildschirm gliedert sich in drei Bereiche:

- das Ausgabe–Fenster
- das List–Fenster
- die Menü–Leiste

Das Ausgabe– und List–Fenster haben einige Eigenschaften gemeinsam:

- Um ein Fenster zu wählen, wird auf eine beliebige Stelle im Innern gezeigt und die Auswahltaste der Maus gedrückt.
- Um die Fenstergröße zu verändern, kann man das Größen–Symbol in der rechten unteren Ecke des Fensterrahmens mit der Maus ziehen.

- Um das Hintergrundfenster in den Vordergrund zu holen, wählt man das Front–Symbol oben rechts im Fensterrahmen.

- Um das Vordergrundfenster in den Hintergrund zu bringen, wählt man das Hintergrund–Symbol oben rechts im Fensterrahmen.

- Um ein Fenster zu schließen, wählt man das Schließ–Symbol in der linken oberen Ecke des Fensterrahmens.

- Um das ganze Fenster nach oben oder unten zu ziehen, zeigt man auf die Titelleiste, hält die Auswahltaste der Maus gedrückt und zieht die Maus.

Die Menü–Leiste bietet eine Reihe unterschiedlicher Merkmale:

- Um die Menü–Leiste anzuzeigen, wählt man das Ausgabe– oder das List-Fenster und hält die Menütaste der Maus gedrückt.

- Zur Anzeige eines individuellen Menüs zeigt man auf den gewünschten Menü–Titel bei weiter gedrückter Menütaste.

- Zur Auswahl aus einem Menü zeigt man mit gedrückter Menütaste auf den gewünschten Menü–Punkt, wodurch dieser hervorgehoben wird. Dann läßt man die Menütaste los.

Das Ausgabe–Fenster

Im Ausgabe–Fenster werden sowohl Anweisungen und Befehle im Direktmodus einegeben als auch Programmausgaben angezeigt.

Das Ausgabe–Fenster wählen Das Auswahlfenster können Sie auf zweierlei Weise wählen:

- Drücken Sie die Auswahltaste der Maus, wenn der Zeiger an beliebiger Position innerhalb des Fensters steht.

- Wählen Sie **Show Output** (Ausgabefenster zeigen) aus dem **Windows**-Menü und drücken Sie dann die Auswahltaste der Maus innerhalb des Fensters.

Arbeiten im Ausgabe–Fenster Im Ausgabe–Fenster können Sie:

- eine Anweisung oder einen Befehl im Direktmodus eingeben und ausführen lassen, sobald die RETURN–Taste gedrückt wird. Außerdem wird jegliche Programmausgabe hier angezeigt.
- die BACKSPACE–Taste verwenden, um Eingabefehler zu korrigieren.
- mit CTRL–C ein Programm anhalten oder eine eingegebene Zeile löschen.

Das List–Fenster

Im List–Fenster wird ein Programm eingegeben, ediert und durchgesehen. Außerdem kann bei Einzelschrittverarbeitung der Programmablauf hier verfolgt werden. Das List–Fenster wird automatisch gewählt, sobald Amiga Basic aufgerufen wird.

Das List–Fenster wählen Das List–Fenster können Sie auf zweierlei Weise wählen:

- Drücken Sie die Auswahltaste der Maus, wenn der Zeiger an beliebiger Position innerhalb des Fensters steht.
- Wählen Sie **Show List** (List–Fenster zeigen) aus dem **Windows**-Menü und drücken Sie dann die Auswahltaste der Maus innerhalb des Fensters.

Wenn das Programm auf Grund eines Fehlers angehalten wird, wird das List–Fenster automatisch gewählt und sichtbar gemacht.

Beachten Sie: Wenn das Programm im geschützten Format gespeichert wurde, können Sie dafür das List–Fenster nicht öffnen. Geschützte Programme können weder gelistet noch ediert werden.

Arbeiten im List–Fenster Im List–Fenster können Sie:

- die Programmliste anschauen und sie mit Hilfe der Cursor–Steuertasten in Verbindung mit der SHIFT– oder ALT–Taste abrollen lassen.
- Eingeben oder Edieren eines Programms unter Zuhilfenahme der komfortablen Edier-Eigenschaften von Amiga Basic. Ausführliches dazu finden Sie in Kapitel 4.

3.4 Die Menü–Leiste und Tastaturabkürzungen für das Menü

Die Menüleiste enthält die Titel für vier Menüs:

- Project (Programme laden und speichern)
- Edit (Programme edieren)
- Run (Programme starten und anhalten)
- Windows (Fenster anzeigen)

Sie können zu jedem Zeitpunkt aus jedem dieser Menüs einen beliebigen Punkt wählen. Ist jedoch ein Menü–Titel oder in einem Menü einer der Punkte in "Geisterschrift", also in undeutlich lesbarer Schrift angegeben, so ist dieses Menü oder dieser Punkt in der gegenwärtige Arbeitssituation nicht verfügbar.

Einige Menüpunkte enthalten rechts neben der Bezeichnung eine Tasten–Kombination aus einer Buchstabentaste sowie der **rechten** Amiga–Taste (das ist die Taste unmittelbar rechts von der Leertaste auf der Tastatur). Ein Beispiel ist die Kombination Amigataste–X–Taste als Ersatz für **Cut** im **Edit**–Menü. Sie müssen in diesem Fall die rechte Amigataste niederhalten und dann die X–Taste drücken. Alle Tastaturabkürzungen verwenden die **rechte** Amigataste.

Das Project–Menü

Das Project–Menü enthält fünf Punkte die sich auf die Arbeit mit Programmdateien beziehen. In diesem Menü gibt es keine Tastaturabkürzungen für die einzelnen Punkte.

New löscht die aktuelle Programmliste im List–Fenster und das Programm im Hauptspeicher, so daß Sie mit der Eingabe eines neuen Programms beginnen können. **New** hat dieselbe Wirkung wie der NEW–Befehl (s. dort in Kapitel 9).

Open veranlaßt Amiga Basic, ein auf Diskette gespeichertes Programm in den Hauptspeicher zu laden. Sie können das Ausgabe–Fenster wählen und den Befehl FILES eingeben, um sich die Dateien auf der Diskette im aktuellen Laufwerk anzuschauen. **Open** fordert in einem Kommunikationsfenster zur Eingabe des Programmnamens auf. Die Eingabe wird durch Wahl des OK-Symboles oder durch Drücken der RETURN–Taste abgeschlossen.

Save speichert das Programm im Hauptspeicher unter seinem aktuellen Namen, nachdem Sie es eingegeben oder geändert haben. Amiga Basic speichert alle neuen Programme im komprimierten Format und alle geänderten Programme in dem Format, in dem sie vorher geladen wurden.

Save As wirkt genauso wie **Save** mit dem Unterschied, daß Sie hier den Namen des zu speichernden Programms vorher ändern können. Auch hier werden neue Programme im komprimierten Format und geänderte Programme in dem Format gespeichert, in dem sie geladen wurden.

Um ein Programm im Text– oder im geschützten Format zu speichern, müssen Sie den SAVE–Befehl im Direktmodus im Ausgabefenster eingeben. Näheres zu den Dateiformaten finden Sie in Kapitel 5 und zu der Syntax des SAVE–Befehls in Kapitel 9.

Quit beendet die Arbeit mit Amiga Basic und führt Sie auf den Arbeitstisch (Workbench) zurück. **Quit** wirkt genauso wie der SYSTEM–Befehl.

Das Edit–Menü

Das Edit–Menü enthält drei Punkte für die Eingabe und Edierung von Programmen. Mit Ausnahme der Eingabe von Anweisungen und Befehlen im Direktmodus im Ausgabe–Fenster erfolgt die Eingabe und Edierung von Programmzeilen ausschließlich im List–Fenster. Für jeden der drei Punkte existiert eine Tastatur–Abkürzung.

Cut löscht den aktuell ausgewählten Bereich aus dem List–Fenster und legt ihn in einem Zwischenspeicher ab. Drücken der Tasten Amiga–X bewirkt dasselbe.

Copy legt den aktuell ausgewählten Bereich im Zwischenspeicher ab, ohne ihn im List–Fenster zu löschen. Drücken der Tasten Amiga–C hat dieselbe Funktion.

Paste ersetzt den aktuell ausgewählten Bereich durch den Inhalt des Zwischenspeichers. Wenn kein Bereich ausgewählt war, wird der Zwischenspeicherinhalt rechts von der Cursor–Position eingefügt. Drücken der Tasten Amiga–P hat dieselbe Wirkung.

Das Run–Menü

Im Run–Menü sind sechs Punkte enthalten, die die Programmausführung steuern. Für vier von ihnen existieren Tastatur–Abkürzungen:

Start startet das aktuell im Hauptspeicher befindliche Programm. Die Eingabe des RUN–Befehls im Ausgabe–Fenster oder Drücken von Amiga–R haben dieselbe Wirkung. Im Direkt–Modus kann ein Amiga Basic–Programm, so im Speicher vorhanden, jederzeit gestartet werden.

Stop unterbricht ein ablaufendes Programm. **Stop** hat dieselbe Wirkung wie die STOP–Anweisung in einem Programm. Die Tastenkombinationen Amiga–. (Punkt) oder CTRL–C sind die Tastaturabkürzungen für **Stop**.

Continue setzt ein unterbrochenes oder angehaltenes Programm fort. Die Eingabe des CONT–Befehls im Ausgabe–Fenster hat dieselbe Wirkung. **Continue** ist nur aktivierbar, wenn ein Programm vorher angehalten oder unterbrochen wurde und die Fortsetzung möglich ist. Wurde kein Programm angehalten oder unterbrochen, oder wurde das Programm während der Unterbrechung verändert, wird in einem Kommunikationsfenster die Meldung **Can't continue** (Fortsetzung nicht möglich) angezeigt.

Suspend hält ein laufendes Programm solange an, bis eine andere Taste als Amiga–S gedrückt wird. Drücken von Amiga–S oder CTRL–S hat dieselbe Wirkung wie die Wahl von **Suspend**. Sobald ein Programm läuft, ist **Suspend** wählbar.

Trace On/Off ist ein Schalter, mit dem die Programmablaufverfolgung zum Zweck der Fehlersuche ein– und ausgeschaltet werden kann. Wenn das List–Fenster sichtbar ist, wird hier jede ausgeführte Anweisung hervorgehoben dargestellt. Wird **Trace On** gewählt, so hat dies dieselbe Wirkung wie der TRON–Befehl, bei dem die letzte ausgeführte Anweisung mit einem Rechteck eingerahmt wird. Wird keine Anweisung ausgeführt, so wird auch kein Rechteck angezeigt. Damit kann bestimmt werden, wo das Programm angehalten wurde. **Trace Off** hat dieselbe Wirkung wie der TROFF–Befehl. Die Hervorhebung der schrittweise abgearbeiteten Anweisungen wird ausgeschaltet.

Step schaltet die schrittweise Programmabarbeitung ein. Nach jeder ausgeführten Anweisung hält das Programm an. Dieselbe Wirkung hat Drücken von Amiga–T. Bei sichtbarem List–Fenster wird die gerade ausgeführte Anweisung mit einem Rechteck umrahmt.

Das Windows–Menü

Das Windows–Menü stellt zwei Punkte für das Öffnen von Fenstern auf dem Amiga Basic–Bildschirm zur Verfügung:

Show List öffnet und wählt das List–Fenster für das aktuelle Programm. Wenn das List–Fenster bereits geöffnet ist, aber vom Ausgabe–Fenster überdeckt wird, wird es mit **Show List** über das Ausgabefenster gelegt. Dieselbe Wirkung hat Drücken von Amiga–L. Zur Edierung eines geladenen oder zur Eingabe eines neuen Programms kann auch der LIST–Befehl im Direktmodus im Ausgabe–Fenster eingegeben werden.

Show Output öffnet das Ausgabe–Fenster und legt ggf. das List–Fenster in den Hintergrund. Um Direktmodus–Befehle oder –Anweisungen im Ausgabe–Fenster eingeben zu können, muß es erst gewählt werden.

4. Edieren und Testen von Amiga–Basic–Programmen

In diesem Kapitel lernen Sie die Erfassung von Amiga Basic–Programmen, ihre Bearbeitung sowie das Testen und die Fehlerbeseitigung.

4.1 Programm–Edierung

Sobald Sie Amiga Basic aufgerufen haben, wird das List–Fenster aktiviert. Zur Eingabe und Bearbeitung der Programmzeilen dienen die schon erwähnten Edier–Hilfen **Cut, Copy** und **Paste** aus dem **Edit**–Menü. Anfangs erscheint das List–Fenster etwas schmal für längere Programmzeilen. Wenn Sie bei der Texteingabe den rechten Rand erreichen, wird der Fensterinhalt nach links gerollt, so daß der Cursor immer im sichtbaren Bereich des Fensters bleibt. Zum linken Rand gelangen Sie wieder, indem Sie ALT–Pfeil nach links drücken. Sie können aber auch durch Ziehen des Fensters nach links und anschließendem Ziehen des Größen–Symbols in der rechten unteren Ecke des Fensterrahmens das List–Fenster über den ganzen Bildschirm legen.

Die Edierung von Programmzeilen im List–Fenster funktioniert prinzipiell wie die Textbearbeitung in einem Text–Programm. Im folgenden werden nocheinmal alle Funktionen des Bildschirmeditors von Amiga Basic zusammengestellt:

Eingeben und Edieren von Text

- Text wird eingefügt, indem er über die Tastatur eingegeben wird oder mit **Paste** aus dem Zwischenspeicher kopiert wird. Text wird immer rechts von der Cursor–Position eingefügt.

- Text wird mit der BACKSPACE–Taste gelöscht oder ausgewählt und dann mit **Cut** aus dem **Edit**–Menü gelöscht.

- Jede Programmzeile wird durch Drücken der RETURN–Taste abgeschlossen. Es sind auch einzelne Leerzeilen in Amiga Basic–Programmen erlaubt. Diese werden bei der Programmausführung ignoriert.

- Textzeilen können mit Hilfe der TAB–Taste eingerückt werden. Drücken der TAB–Taste setzt den Cursor um drei Schreibpositionen nach rechts. Wird am Zeilenende die RETURN–Taste gedrückt, so wird der Cursor in der nächsten Zeile direkt unterhalb dem ersten Zeichen in der vorangegangenen Zeile positioniert. Beginnt die vorangegangene Zeile mit einem Tabulator, so beginnt die Folgezeile automatisch ebenfalls mit einem Tabulator. Dieses Einrücken von Text kostet **keinen** zusätzlichen Speicherplatz.

- Reservierte Wörter, also die Schlüsselwörter der Amiga Basic–Sprache, können in Klein–, Groß– oder gemischter Schreibweise eingegeben werden. Amiga Basic wandelt sie generell in Großschrift und zeigt sie auch so an.

- Variablennamen dürfen bis zu 40 Zeichen lang sein. Numerische Variablen werden voreinstellungsgemäß immer als einfachgenaue Variablen angenommen, es sei denn, Sie geben durch ein spezielles Zeichen etwas anderes an oder verwenden DEFINT–, DEFLNG–, DEFSNG–, DEFDBL– oder DEFSTR–Anweisungen. Die speziellen Typ–Zeichen sind **$** für Zeichenkette, **!** für einfache Genauigkeit, **#** für doppelte Genauigkeit, **%** für kurze und **&** für lange Ganzzahl. Variablennamen können in Klein–, Groß– oder gemischter Schreibweise eingegeben werden. Intern werden sie generell in Großschreibweise gewandelt.

  Alpha, ALPHA, AlpHa und alpha sind also dasselbe.

- Programmzeilen können Nummern vorangestellt werden. Sie sind jedoch nicht erforderlich und die Zeilen mit Nummern werden **nicht** nach aufsteigenden Nummern geordnet.

Text auswählen

- Zeichen, Wörter oder ganze Zeilen können ausgewählt werden, indem die Hervorhebung mit der Maus gezogen wird.

- Die schnellste Methode, eine ganze Zeile auszuwählen, ist, auf den Zeilenanfang zu zeigen und die Hervorhebung eine Zeile nach unten zu ziehen.

- Wird die Hervorhebung über den rechten Rand des List–Fensters hinausgezogen, so wird der Fensterinhalt kontinuierlich mit dem Ziehen nach links gerollt.

- Einzelne Wörter in verschiedenen Zeilen können durch Zeigen und zweimaliges kurzes Drücken der Auswahltaste der Maus ausgewählt werden.

Eine weitere Möglichkeit, einen größeren Textblock auszuwählen besteht darin, am Blockanfang die Auswahltaste der Maus einmal kurz zu drücken, dann auf das Ende des Textes zu zeigen, und diesmal bei gedrückter SHIFT–Taste die Auswahltaste der Maus nocheinmal zu drücken.

Text im List–Fenster rollen

- Wenn Sie bei der Texteingabe das untere Ende des List–Fenster erreicht haben und weiter eingeben, wird der Text im Fenster zeilenweise nach oben gerollt.

- Wenn Sie bei der Texteingabe den rechten Rand des List–Fensters erreichen und weiter eingeben, wird der Fensterinhalt kontinuierlich nach links gerollt.

- Der Text–Cursor, der gleichzeitig die Position markiert, bei der Text ein– oder angefügt wird, kann mit den Cursor–Steuertasten (den vier Pfeiltasten) nach rechts, links, oben oder unten bewegt werden.

- Wenn Sie die Pfeil–nach–rechts–Taste drücken und der Cursor am rechten Rand des List–Fensters steht, wird der Fensterinhalt um ca. 75% der Fensterbreite nach links gerollt. Das äußerst rechte Ende der möglichen Anzeige wird durch einen Piepton angezeigt. Entsprechendes gilt für die anderen drei Pfeiltasten.

- Wenn Sie bei niedergehaltener SHIFT–Taste eine der Pfeiltasten drücken, wird die Anzeige in die entsprechende Richtung gerollt. Am Ende der jeweiligen Richtung wird ein Piepton erzeugt.

- Zum fensterweisen Durchsehen eines Programms zum Programmende hin dient die Tastenkombination SHIFT–Pfeil nach unten, zum Programmanfang hin SHIFT–Pfeil nach oben.

- Um zum Anfang einer Programmliste zu gelangen, dient die Tastenkombination ALT–Pfeil nach oben, zum Programmende ALT–Pfeil nach unten.

- Um zum Ende der aktuellen Programmzeile zu gelangen, dient die Tastenkombination ALT–Pfeil nach rechts, zum Zeilenanfang ALT–Pfeil nach links.

- Um ca. 75% der Länge der aktuellen Programmzeile zum Zeilenende hin zu überspringen, dient die Tastenkombination SHIFT–Pfeil nach rechts, zum Zeilenanfang hin SHIFT–Pfeil nach links.

List–Fenster bei einer bestimmten Zeile oder Marke öffnen

Um das List–Fenster und damit die Programmanzeige bei einer bestimmten Zeile zu öffnen, geben Sie im Ausgabe–Fenster den Befehl LIST und die entsprechende Zeilennummer oder Sprungmarke ein. Das List–Fenster wird dann geöffnet und zeigt als erste Zeile die angegebene Zeile an.

Bei dem in Kapitel 2 behandelten Beispielprogramm Picture wird z.B. durch die Eingabe

```
LIST MovePicture
```

das List–Fenster geöffnet und die Programmliste beginnend mit dem MovePicture–Unterprogramm gezeigt.

4.2 Testen von Amiga Basic–Programmen

Im folgenden werden die vier Hilfsmittel beschrieben, die Amiga Basic für das Testen und die Fehlerbeseitigung bei Basic–Programmen zur Verfügung stellt. Damit wird die Fehlersuche und –beseitigung wesentlich erleichtert.

Fehlermeldungen

Wenn Amiga Basic bei der Programmabarbeitung einen Fehler findet, wird das Programm angehalten und ein Kommunikationsfenster wird geöffnet, in dem eine Fehlermeldung angezeigt wird. Die fehlerhafte Zeile selbst wird im List–Fenster angezeigt, wenn dieses sichtbar ist. Im Anhang B sind alle Fehlermeldungen zusammen mit den Fehlercodes tabellarisch zuammengefaßt. Dort werden auch Hinweise der möglichen Fehlerursache und –Beseitigung gegeben.

Der TRON–Befehl

Der TRON–Befehl hat seinen Namen von **TR**ace **ON** (Programmablaufverfolgung ein). Dieser Modus wird durch Wählen von **Trace** aus dem **Run**–Menü, durch einen TRACE–Befehl in einem Programm oder durch Eingeben von TRACE im Ausgabe–Fenster eingeschaltet.

Wenn das List–Fenster sichtbar ist, wird jede Anweisung, die gerade ausgeführt wird, mit einem orangefarbenen Rechteck umrahmt.

Um die Programmablaufverfolgung wieder auszuschalten, wählen Sie **Trace** aus dem **Run**–Menü erneut, führen einen TRACE OFF–Befehl in einem Programm aus oder geben im Ausgabe–Fenster TRACE OFF ein.

Wenn Sie einen Programmfehler auf einen kleinen Programmbereich eingegrenzt haben, ist es leichter und schneller, den TRON–Modus vom Programm aus einzuschalten, kurz ehe der Fehler auftritt.

Schrittweises Abarbeiten von Programmen

Mit Hilfe von **Step** aus dem **Run**–Menü oder durch Drücken von Amiga–T kann ein Programm schrittweise abgearbeitet werden. Wenn ein Programm unterbrochen wurde, wird mit **Step** die nächste Anweisung im Programm ausgeführt und das Programm wird dann erneut unterbrochen. Bei mehreren Anweisungen in der Zeile wird jede Anweisung einzeln ausgeführt.

Ist das List–Fenster sichtbar, wird die zuletzt ausgeführte Anweisung umrahmt.

Sie können sich so schrittweise durch ein Programm hindurcharbeiten und zu jeder Zeit die Inhalte von Programmvariablen anschauen, indem Sie die PRINT–Anweisung im Ausgabe–Fenster verwenden.

Wenn Sie die END–Anweisung im Ausgabe–Fenster eingeben, wird **Step** ausgeschaltet und das Programm kann von Anfang an neu gestartet werden.

Bei aktivierter Unterbrechungsreaktionsfähigkeit (BREAK ON–Anweisung, s. dort in Kapitel 9) kann **Step** nicht verwendet werden.

Pausen beim Programmablauf

Ein ablaufendes Programm kann mit **Suspend** aus dem **Run**–Menü oder durch Drücken von Amiga–S angehalten werden, bis eine beliebige andere Taste außer Amiga–S gedrückt oder **Continue** aus dem **Run**–Menü gewählt wird. **Suspend** ist immer dann aufrufbar, wenn ein Programm abläuft.

Fortsetzen unterbrochener Programme

Unterbrochene oder angehaltene Programme können entweder durch Eingabe des CONT–Befehls im Ausgabe–Fenster oder mit **Continue** aus dem **Run**–Menü fortgesetzt werden.

Verwendung der Edier–Hilfen Cut, Copy und Paste im List–Fenster

Vergessen Sie nicht, daß der Inhalt des Editor–Zwischenspeichers bei jedem Aufruf der Edier–Hilfen **Cut** oder **Copy** überschrieben wird. **Paste** dagegen verändert den Zwischenpufferinhalt nicht, da diese Edier–Hilfe nur lesend zugreift. Deshalb kann mit **Paste** der Zwischenpufferinhalt so oft in verschiedene Programmteile eingefügt werden, wie notwendig.

Es kann vorkommen, daß man einen Programmteil löschen möchte, ohne den Zwischenspeicher zu überschreiben. Dies kann folgendermaßen erreicht werden: Sie wählen zunächst den zu löschenden Teil aus, der dadurch im List–Fenster hervorgehoben erscheint. Anschließend löschen Sie ihn durch wiederholtes Drücken der BACKSPACE–Taste. Auf diese Weise verhindern Sie auch die Fehlermeldung **Out of Heap Space** (Arbeitsspeicher–Überlauf), die dann wahrscheinlich wird, wenn sehr große Textblöcke mit **Cut** gelöscht werden.

4.3 Programmtest mit dem Ausgabe–Fenster

Wenn ein Programm angehalten oder unterbrochen wurde, kann man sich mit Hilfe des Ausgabe–Fensters und des Direktmodus nützliche Informationen über den augenblicklichen Programmzustand anzeigen lassen. Wenn das Programm z.B. mit einer Fehlermeldung abgebrochen wurde, als es gerade

eine Schleife abgearbeitet hat, kann man sich die Anzahl der bereits ausgeführten Schleifendurchläufe zusammen mit den Werten aller Variablen anzeigen lassen. Dazu wird die PRINT–Anweisung (s. dort in Kapitel 9) im Direktmodus im Ausgabe–Fenster eingegeben.

Eine andere Verwendungsmöglichkeit des Ausgabe–Fensters beim Programmtest besteht im Verändern von Variablenwerten mit Hilfe der LET–Anweisung (s. dort in Kapitel 9) im Direktmodus. Sie können einer Variablen des unterbrochenen Programms einen neuen Wert zuweisen, und mit **Continue** aus dem **Run**–Menü die Verarbeitung fortsetzen.

5. Arbeiten mit Dateien und Geräten

In diesem Kapitel wird beschrieben, wie mit Amiga Basic Ein–/Ausgabeinformationen verarbeitet werden und wie der Interpreter Dateien und Geräte behandelt. Weiterhin wird die Dateiverwaltung mit Amiga Basic behandelt und es werden Möglichkeiten des Datentransfers zwischen Amiga Basic und Textverarbeitungsprogrammen diskutiert.

5.1 Allgemeine Geräte–Ein–/Ausgabe

Amiga Basic unterstützt verallgemeinerte Ein–/Ausgabe. Das hat den Vorteil, daß verschiedene Ein–/Ausgabegeräte mit derselben Syntax programmiert werden können, die Amiga Basic für den Zugriff auf Disk–Dateien (Floppy–Disk oder Festplatte) verwendet. Dabei kennt Amiga Basic folgende Geräte mit den Namen:

SCRN: Ausgabedateien können unter diesem Namen geöffnet werden, wenn Daten an den Bildschirm ausgegeben werden sollen.

KYBD: Eingabedateien können unter diesem Namen geöffnet werden, wenn die Daten von der Tastatur gelesen werden sollen.

LPT1: Ausgabedateien können unter diesem Namen geöffnet werden, wenn Daten an den Drucker ausgegeben werden sollen. Wird **LPT1:BIN** als Namen verwendet, so werden die Daten im Binärformat ausgegeben. In diesem Fall werden die Daten von Amiga Basic nicht interpretiert, also keine Leerstellen bei Tabulatorcodes und keine Wagenrücklaufcodes beim Überschreiten der eingestellten Zeilenlänge ausgegeben.

COM1: Ein–/Ausgabedateien können unter diesem Namen geöffnet werden, wenn Daten über die serielle Schnittstelle (RS232–Port) ein– oder ausgegeben werden sollen. Amiga Basic erkennt die folgenden Parameter als Bestandteil des **COM1:**–Dateinamens:

COM1:[***Baud***][,[***Parität***][,[***Wortl***][,[***Stopb***]]]]

Baud Die Übertragungsgeschwindigkeit in Bits/Sekunde als Zahlenwert. Die hiermit gesetzte Baud–Rate hat Priorität vor der mit dem Voreinsteller Preferences gewählten. Folgende Baud–Raten sind möglich: 110, 150, 300, 600, 1200, 1800, 2400, 3600, 4800, 7200, 9600, 19200.

Parität Ein Buchstabe, der die Art der Paritätsprüfung der übertragenen Daten angibt. Möglich sind hier **E** für gerade Parität (Even), **O** für ungerade Parität (Odd) oder **N** für keine Paritätsprüfung (None).

Wortl Eine Ziffer, die angibt, aus wievielen gültigen Datenbits ein Datenwort besteht. Erlaubt sind hier 5, 6, 7 oder 8.

Stopb Eine Ziffer, die angibt, wieviele Bits das Ende eines Datenwortes markieren. Bei 110 Baud sind es standardmäßig 2, bei allen anderen Baudraten standardmäßig 1. Bei 5 Datenbits und angegebenen 2 Stopbits werden intern 1.5 Stopbits verwendet.

Beispiel:

```
OPEN "COM1:300,N,7,2" AS #1
```

S. a. OPEN "COM1:–Anweisung in Kapitel 9.

Drucker–Unterstützung

Der Amiga unterstützt eine ganze Reihe von Druckern, die beim Voreinsteller Preferences angegeben sind. Soll die von Ihnen erzeugte Druckerausgabe Funktionen wie Randsetzen, Kursivschrift und ähnliches benutzen, müssen Sie dazu spezielle Druckercodes ausgeben.

Für diesen Zweck enthält der Amiga ein spezielles Druckertreiber–Programm für jeden unterstützten Drucker. Jedes dieser Programme wandelt die Standard–Druckersteuerzeichen in die speziellen Zeichenfolgen für den jeweiligen Drucker.

AmigaDOS unterscheidet drei Drucker–Einheiten:

- PRT:
- SER:
- PAR:

Bei SER: und PAR: werden die Daten über die serielle bzw. parallele Schnittstelle ausgegeben. Hier werden jedoch die Standard–Drucksteuerzeichen nicht umgesetzt, so daß diese beiden Einheiten für die normale Anwendung ziemlich ungeeignet sind. Bei seriellen Übertragungen (z.B. Terminal–Emulatoren oder Datentransfer zu anderen Computern ist die COM1:–Einheit besser geeignet als SER:, weil mit ihr direkt die Übertragungsparameter wie Baud–Rate, Parität und Wortlänge gesetzt werden können.

Die PRT:–Einheit arbeitet bevorzugt mit dem bereits weiter oben erwähnten Gerät LPT1:. LPT1: ist ein von Microsoft vergebener Gerätename, um die Software–Übertragbarkeit zwischen unterschiedlichen Rechnern zu gewährleisten.

Wenn Sie Ihre Programmausgabe speziell formatieren wollen, können Sie die entsprechenden Drucker–Steuercodes in PRINT#–Anweisungen im Programm angegeben. Diese sogenannten "Escape–Sequenzen" bestehen aus dem ESC–Zeichen (ASCII–Code 27), gefolgt von einem oder mehreren anderen Zeichen.

Wenn Sie z.B. den Commodore–Drucker MPS–1000 an Ihrem Amiga angeschlossen haben und bestimmte Stellen in Ihrem Dokument durch Unterstreichung hervorheben wollen, gehen Sie folgendermaßen vor:

Wählen Sie zunächst diesen Druckertyp mit dem Voreinsteller Preferences (s. Amiga–Benutzerhandbuch). Dann geben Sie die Escape–Sequenzen zum Ein– und Ausschalten der Unterstreichungs–Funktion in PRINT#–Anweisungen an, wie es das folgende Beispiel demonstriert:

```
UnterEIN$=CHR$(27)+"[4m"
UnterAUS$=CHR$(27)+"[24m"
Text1$="Normaler Text"
Text2$="Unterstrichener Text"

OPEN "LPT1:" FOR OUTPUT AS #2
    PRINT#2,Text1$
    PRINT#2,UnterEIN$+Text2$
    PRINT#2,UnterAUS$+Text1$
CLOSE #2
```

5.2 Dateibenennungs–Konventionen

Amiga Basic hat einige Vorschriften hinsichtlich der Angabe von Dateinamen. Jede Datei wird durch ihren Namen mit optional vorangestellter Disk–Kennung und/oder einem oder mehreren geschachtelten Unterverzeichnis–Namen beschrieben. Eine komplette Dateibezeichnung wird als **Pfadname** bezeichnet.

Dateinamen

Amiga Basic unterstützt die hierarchische Dateiorganisation des AmigaDOS, bei der Unterverzeichnisse erlaubt sind, vollständig. Es kennt also auch Pfadnamen (s. dazu AmigaDOS–Benutzerhandbuch, Kapitel 1.3).

Ein kompletter Pfadname (das ist die Angabe, mit deren Hilfe Amiga Basic eine bestimmte Datei im hierarchisch gegliederten Dateisystem des AmigaDOS findet) darf zwischen 1 und 255 Zeichen lang sein und kann sowohl aus Groß– und Kleinbuchstaben als auch aus einem Gemisch aus beiden bestehen. Jeder Datei– oder Unterverzeichnisname innerhalb eines Pfadnamens darf bis zu 30 Zeichen lang sein, darf jedoch keine Steuerzeichen enthalten. Gültige Dateinamen sind z.B.:

```
KUNDEN A2400 MeineDatei TEST DATEN
```

Um ein bestimmtes Laufwerk oder eine Diskkennung als Bestandteil eines Pfadnamens anzugeben, wird dessen/deren Namen, gefolgt von einem Doppelpunkt, vor dem Dateinamen angegeben, also z.B.:

```
Demos:Bilder
DF1:KontenARCHIV
```

Ein Unterverzeichnis–Name (wie z.B. eine Schublade auf dem Arbeitstisch) wird mit einem Schrägstrich (/) vom eigentlichen Dateinamen getrennt, also z.B.:

```
BasicDemos/Picture
Briefe:Notizen/arbeitsdatei
DF1:Berichte/März
```

in den beiden letzten Beispielen wird gezeigt, wie eine Diskkennung vor dem Unterverzeichnisnamen angegeben wird. Weitere Informationen zu Dateinamen finden Sie im AmigaDOS–Benutzerhandbuch, Kapitel 1.3.

Disk–Kennung

Ihr Amiga ist mit einem eingebauten Floppy–Disk–Laufwerk ausgerüstet. Zum Ausbau der externen Speicherkapazität lassen sich weitere Laufwerke anschließen. Aber gerade bei nur einem Laufwerk werden Sie sicher häufig mit mehreren Datendisketten arbeiten. In diesem Fall müssen Sie bei einer Dateieröffnung angeben, auf welche Diskette zugegriffen werden soll. Zu diesem Zweck wird bei der Formatierung einer Diskette ein Name, die Disk–Kennung, vergeben. Diese Kennung stellen Sie bei der Eröffnung einer Datei mit einem Doppelpunkt an den Anfang des Pfadnamens.

Das Laden von Programmdateien geht am einfachsten mit **Open** aus dem **Project**–Menü. Befindet sich das zu ladende Programm auf einer anderen Diskette, nehmen Sie die Diskette aus dem eingebauten Laufwerk und legen Sie die entsprechende Diskette hier ein. Danach können Sie sich mit dem FILES–Befehl (s. dort in Kapitel 9) den Disketteninhalt ansehen und dann die gewünschte Programmdatei in der üblichen Weise laden. Wenn Sie eine Datei auf einer anderen als der eingelegten Diskette speichern wollen, wählen Sie am besten **Save As** aus dem **Project**–Menü und verfahren dann genau wie beim Laden.

Das Laden von anderen Disketten kann ebenso mit den LOAD–, MERGE– oder RUN–Befehlen (s. dort in Kapitel 9) geschehen, indem Sie bei der Eingabe im Direktmodus im Ausgabefenster vor den Dateinamen die Disk-Kennung zusammen mit einem Doppelpunkt angeben. Wenn jedoch nicht vorher schon die richtige Diskette eingelegt wurde, wird die Fehlermeldung **Unknown Volume** (unbekannte Diskette) angezeigt.

5.3 Das Arbeiten mit Dateien

In diesem Abschnitt werden die grundlegenden Ein–/Ausgabe–Mechanismen für den Amiga Basic–Anfänger beschrieben. Aber auch der mit einem dateibezogenen Fehler Konfrontierte kann hier nachschlagen, um sich über die korrekte Anwendung der Dateibearbeitungsanweisungen zu informieren.

Befehle und Anweisungen für Programmdateien

Im folgenden erhalten Sie einen kurzen Überblick über die Befehle und Anweisungen, mit denen Sie Programmdateien verwalten können. Detaillierte Hinweise und Syntaxbeschreibungen finden Sie bei den jeweiligen Befehlen oder Anweisungen in Kapitel 9.

Eröffnen einer Programmdatei

Eine Programmdatei können Sie auf drei verschiedene Weisen öffnen. Die üblichste ist der LOAD–Befehl. Wenn Sie eine Programmdatei laden, werden alle ggf. zu diesem Zeitpunkt geöffneten Dateien geschlossen, der Hauptspeicher wird gelöscht und das Programm wird in den Hauptspeicher geladen.

Eine andere Möglichkeit zum Laden einer Programmdatei besteht im Anfügen der geladenen Datei an ein bereits im Hauptspeicher befindliches Programm. Dies geschieht mit Hilfe des MERGE–Befehls (s. dort in Kapitel 9) und ist immer dann zu empfehlen, wenn ein sehr großes Programm erstellt wird, das in einzelnen Segmenten getestet wird. Im Anschluß an das Testen können dann die einzelnen Segmente zusammengemischt werden.

Eine dritte Möglichkeit besteht in der Übergabe der Steuerung an ein anderes Programm innerhalb eines Programmes mit Hilfe der CHAIN–Anweisung (s. dort in Kapitel 9). Wird diese Anweisung verwendet, so lädt ein im Hauptspeicher befindliches Programm ein anderes Programm. Das aufrufende Programm wird dann überlagert. Wahlweise Parameter erlauben die Übergabe bestimmter oder aller Variablen oder auch das Einmischen des geladenen in das aufrufende Programm.

Speichern in Programmdateien

Üblicherweise wird eine Programmdatei mit Hilfe von **Save** oder **Save As** aus dem **Project**–Menü (s. Kapitel 3.4) oder durch Eingeben des **SAVE**–Befehls (s. dort in Kapitel 9) auf Diskette gespeichert. Voreinstellungsmäßig werden Programmdateien im Binärformat gespeichert.

Wenn Sie die Programmdatei im geschützten Format speichern wollen, um Listen oder Änderungen zu verhindern, können Sie den Parameter **P** beim SAVE–Befehl verwenden. Zusätzlich sollten Sie aber immer eine ungeschützte Version speichern, um Änderungen ausführen zu können.

Mit dem Parameter **A** können Sie mit dem SAVE–Befehl ein Programm im ASCII–Format speichern, also in dem Format, in dem es im List–Fenster angezeigt oder auf dem Drucker ausgedruckt wird. Eine ASCII–Datei benötigt zwar mehr Platz auf der Diskette, kann andererseits aber von Textprogrammen gelesen werden. Außerdem verlangen die Befehle CHAIN MERGE und MERGE dieses Format.

Weitere Datei–Befehle

Der NAME–Befehl erlaubt die Umbenennung beliebiger Programm– oder Datendateien auf Diskette. Mit dem KILL–Befehl können Sie Programm– oder Datendateien von Diskette löschen. Nähere Informationen zu diesen beiden Befehlen finden Sie in Kapitel 9.

5.4 Datendateien für sequentiellen und direkten Zugriff

Amiga Basic erlaubt die Erzeugung und Bearbeitung von zwei verschiedenen Typen von Daten–Dateien:

- Sequentielle Dateien
- Direktzugriffs–Dateien.

Sequentielle Dateien

Sequentielle Dateien sind zwar einfacher zu erzeugen als Direkzugriffs–Dateien, sind jedoch nicht so flexibel und erlauben auch nicht einen so schnellen Datenzugriff wie letztere. In einer sequentiellen Datei werden die Datenelemente als Gruppen von ASCII–Zeichen eines nach dem anderen (sequentiell) in der Datei abgelegt. In derselben Ordnung werden sie dann auch wieder aus der Datei gelesen.

Achtung: Eine sequentielle Datei kann entweder nur zum Schreiben oder nur zum Lesen geöffnet werden. Nicht für beide Zugriffsarten gleichzeitig. Wenn Sie an eine bestehende und schon einmal geschlossene sequentielle Datei Daten anfügen wollen, dürfen Sie diese Datei nicht einfach zum Schreiben

öffnen. Das würde nämlich die vorher dort gespeicherten Daten zerstören. Verwenden Sie in diesem Fall den Anfüge–Modus (s. OPEN–Anweisung in Kapitel 9).

Amiga Basic erlaubt die Angabe einer bestimmten Dateipuffergröße bei sequentiellen Dateien. Voreingestellt sind hier 128 Bytes. Die Puffergröße wird in der OPEN–Anweisung angegeben. Sie ist unabhängig von der Länge der zu lesenden oder zu schreibenden Datensätze und hat nur Einfluß auf die Verarbeitungsgeschwindigkeit. Größere Puffer erhöhen sie, verbrauchen andererseits aber auch mehr Hauptspeicherplatz.

Die folgenden Anweisungen und Funktionen werden bei der Arbeit mit sequentiellen Dateien verwendet:

CLOSE	OPEN
EOF	PRINT
INPUT#	PRINT#
INPUT$	PRINT# USING
LINE INPUT#	WIDTH
LOC	WRITE
LOF	

Eine sequentielle Datei erzeugen Das unten angegebene kleine Beispielprogramm zeigt, wie mit Amiga Basic eine sequentielle Datei mit Namen DATEN erzeugt wird und mit Daten, die über die Tastatur eingebene werden, gefüllt wird.

Programm 1: Erzeugen einer sequentiellen Datei

```
OPEN "DATEN" FOR OUTPUT AS #1
Eingabe:
    INPUT "Name ('ENDE' zum Beenden)";N$
    IF N$="ENDE" THEN GOTO Fertig
    INPUT "Abteilung";ABT$
    INPUT "Einstellungsdatum";EINDAT$
    WRITE #1,N$,ABT$,EINDAT$
    PRINT
    GOTO Eingabe
Fertig:
    CLOSE #1
    END
```

Wie im Beispiel gezeigt, werden folgende Programmschritte benötig, um eine sequentielle Date zu erzeugen und Daten abzulegen:

1. Die Datei für Ausgabe öffnen.

2. Daten mit der WRITE #–Anweisung in die Datei schreiben.

3. Am Ende der Datenübertragung die Datei schließen.

Mit Hilfe der PRINT # USING–Anweisung können die Daten für die Ausgabe formatiert werden. Z.B. können mit der Anweisung

```
PRINT#1, USING"####.##,";A,B,C,D
```

numerische Daten in die Datei geschrieben werden, wobei die einzelnen Datenelemente mit Kommata voneinander getrennt werden. Dies geschieht durch die Angabe des Kommas am Ende der USING–Formatzeichenkette. Es sollten grundsätzlich Trennzeichen zum Trennen von Datenelementen verwendet werden, um das Lesen der Daten zu vereinfachen.

Wenn Sie Kommata als Trennzeichen zwischen Datenelementen einfügen wollen, ohne sie extra anzugeben, können Sie auch die WRITE #–Anweisung verwenden. Z.B.

```
WRITE #1,A,B$
```

Hier wird zwischen den Werten der beiden Variablen ein Komma in die Datei eingefügt.

Daten aus einer sequentiellen Datei lesen In einem weiteren Beispielprogramm sehen Sie, wie die Daten aus der Datei DATEN, die mit dem ersten Beispielprogramm erzeugt wurde, gelesen werden. Es sollen aber nur die Namen der Beschäftigten angezeigt werden, die im Jahre 1985 eingestellt wurden.

Programm 2: Aus einer sequentielle Datei lesen

```
OPEN "I", #1,"DATEN"
WHILE NOT EOF(1)
    INPUT #1,N$,ABT$,EINDAT$
    IF RIGHT$(EINDAT$,2)="85" THEN PRINT N$
WEND
```

Es wird jedes Datenelement sequentiell gelesen und daraufhin untersucht, ob am Ende die Zahl 85 steht. Nur dann wird der Name angezeigt. Die Steuerschleife WHILE. . .WEND verwendet vor jedem Lesevorgang die EOF–Funktion, um auf das Dateiende zu prüfen. Damit wird verhindert, daß ein Fehler auftritt, wenn versucht wird, über das Dateiende hinaus zu lesen.

Daten an eine sequentielle Datei anfügen Einer existierenden sequentiellen Datei können nicht einfach Daten hinzugefügt werden, indem die Datei für Ausgabe geöffnet wird. Dies würde nämlich die vorhandenen Daten zerstören. Statt dessen müssen Sie den Anfüge–Modus verwenden. Falls die Datei nicht vorher existiert hat, arbeitet der Anfügemodus genau wie der normale Ausgabemodus.

Programm 3: Daten sequentiell anfügen

```
OPEN "A",#1,"PERSONEN"
REM *** Hinzufügen neuer Personendaten
Neueingabe:
    INPUT "Name";N$
    IF N$="" THEN GOTO Fertig ' RETURN zum Beenden
    LINE INPUT "Adresse? ";ADR$
    LINE INPUT "Geburtstag? ";GEB$
    PRINT #1,N$
    PRINT #1,ADR$
    PRINT #1,GEB$
    GOTO Neueingabe
Fertig:
    CLOSE #1
    END
```

Zur Eingabe der Adresse oder des Geburtstages wird die LINE INPUT–Anweisung verwendet, weil bei ihr Trennzeichen wie z.B. Kommata als zu den Eingabedaten gehörig akzeptiert werden.

Direktzugriffs–Dateien

Die Erzeugung und der Datenzugriff bei Direktzugriffsdateien benötigen mehr Programmmschritte als jene bei sequentiellen Dateien. Die Verwendung von Direktzugriffsdateien hat jedoch einige gravierende Vorteile. So benötigen diese Dateien weniger Speicherplatz auf der Diskette, da numerische Daten im gepackten Binärformat gespeichert werden, bei sequentiellen Dateien jedoch grundsätzlich als Folge von ASCII–Zeichen.

Der größte Vorteil der Direktzugriffsdateien gegenüber den sequentiellen Dateien besteht aber darin, daß auf jeden beliebigen Datensatz der Datei direkt zugegriffen werden kann. Es ist also nicht notwendig, erst alle Informationen vom Dateianfang her zu überlesen, bis der gewünschte Datensatz erreicht ist, wie dies bei sequentiellen Dateien erforderlich ist. Dies wird dadurch ermöglicht, daß die Information in numerierten Datensätzen gespeichert wird. Die Satznummer stellt also die Beziehung zwischen der Information selbst und deren Platz auf der Diskette her.

Folgende Anweisungen und Funktionen werden bei der Arbeit mit Direkzugriffsdateien verwendet:

CLOSE	GET	MKL$
CVD	LOC	MKS$
CVI	LOF	OPEN
CVL	LSET	PUT
CVS	MKD$	RSET
FIELD	MKI$	

Eine Direktzugriffsdatei erzeugen Wie bei den sequentiellen Dateien soll auch hier an einem Beispielprogramm gezeigt werden, wie eine Direktzugriffsdatei erzeugt wird und wie Daten hineingeschrieben werden.

Programm 4: Erzeugung einer Direktzugriffsdatei

```
OPEN "R",#1,"DATEN",32
FIELD #1,20 AS N$,4 AS B$,8 AS T$
Start:
    INPUT "Zweistell. Personalnr. (-1 für Ende)";CODE%
    IF CODE%=-1 THEN Fertig
    INPUT "Name: ";PERSON$
    INPUT "Betrag: ";BETR
    INPUT "Tel.: ";TEL$
    PRINT
    LSET N$=PERSON$
    LSET B$=MKS$(BETR)
    LSET T$=TEL$
    PUT #1,CODE%
    GOTO Start
Fertig:
    CLOSE #1
    END
```

Die Erzeugung einer Direktzugriffsdatei mit Daten erfolgt also mit folgende Programmschritten:

1. Die Datei wird für direkten Zugriff ("R" = Random access) geöffnet. Wird nicht, wie im Beispiel geschehen, die Satzlänge explizit angegeben, so wird eine Länge von 128 Bytes angenommen.

2. Der Datenpuffer für die Datei wird mit der FIELD–Anweisung in Felder unterteilt, damit die zu speichernden Werte der einzelnen Variablen bei jedem Datensatz an eine definierte Stelle gespeichert werden. Über diesen Puffer werden die Programmdaten in die Datei geschrieben oder aus ihr gelesen. Im obigen Beispiel also:

```
FIELD #1,20 AS N$
```

3. Mit der LSET–Anweisung werden die zu speichernden Daten in den Puffer übertragen. Numerische Werte müssen dabei mit Hilfe z.B. der MKS$–Funktion als Zeichenkette interpretiert werden, also gewissermaßen als binäre Zeichenkette. Für die verschiedenen Darstellungsweisen numerischer Werte gibt es verschiedene MKx$–Funktionen. MKS$ dient zur Interpretation von einfachgenauen Gleitpunktwerten.

   Im obigen Beispiel waren die zu speichernden Daten:

```
LSET N$=PERSON$
LSET B$=MKS$(BETR)
LSET T$=TEL$
```

4. Um die Daten aus dem Puffer auf die Diskette zu schreiben, benutzt man die PUT–Anweisung, bei der man auch eine Satznummer angeben kann. Im obigen Beispiel:

```
PUT #1,CODE%
```

In dem Beispielprogramm 4 werden also Informationen über die Tastatur eingeben und in einer Direktzugriffsdatei gespeichert. Die zweistellige Personalnummer, die in der Ganzzahlvariablen CODE% gespeichert wird, dient als Satznummer für die PUT–Anweisung.

Achtung: Variablennamen, die Sie in einer FIELD–Anweisung zur Aufteilung des Datenpuffers in Felder verwendet haben, dürfen Sie nicht z.B. in einer INPUT– oder LET–Anweisung verwenden. Andernfalls wird die Zuordnung

zum Datenpuffer aufgehoben und die Variable wird wie eine normale Programmvariable behandelt.

Daten aus einer Direkzugriffsdatei lesen Im nächsten Beispielprogramm wird gezeigt, wie aus der mit Programm 4 erzeugten Direktzugriffsdatei beliebige Datensätze gezielt gelesen werden können.

Programm 5: Lesen aus einer Direktzugriffsdatei

```
OPEN "R",#1,"DATEN",32
FIELD #1,20 AS N$,4 AS B$,8 AS T$
Start:
    INPUT "Zweistell. Personalnr. (-1 für Ende)";CODE%
    IF CODE%=-1 THEN Fertig
    GET #1,CODE%
    PRINT N$
    PRINT USING "DM#####.##";CVS(B$)
    PRINT T$:PRINT
    GOTO Start
Fertig:
    CLOSE #1
    END
```

Der Zugriff auf Daten in einer Direktzugriffsdatei erfolgt also mit folgenden Programmschritten:

1. Die Datei wird für direkten Zugriff eröffnet.

2. Der Datenpuffer für die Datei wird mit der FIELD–Anweisung in Felder unterteilt, damit die zu lesenden Werte einzelnen Variablen zugewiesen werden können. Weitere Einzelheiten zur FIELD–Anweisung finden Sie bei der Erklärung des Programmbeispiels 4 weiter oben sowie in Kapitel 9.

   Achtung: Bei vielen Programmen, die Daten–Ein– und –Ausgabe für ein und dieselbe Direktzugriffsdatei enthalten, genügt meistens eine OPEN– und eine FIELD–Anweisung.

3. Um den gewünschten Datensatz in den Datenpuffer zu lesen, wird die GET–Anweisung verwendet.

Sobald die Daten im Datenpuffer stehen, kann das Programm über die in der FIELD–Anweisung verwendeten Variablen mit den Daten arbeiten. Numerische Werte, die mit Hilfe der MKx$–Funktionen für die Speicherung als binäre Zeichenketten interpretiert wurden, können jetzt umgekehrt mit Hilfe der

CVx–Funktionen wieder als numerische Werte behandelt werden. Im obigen Beispiel wird der im Datenpuffer als binäre Zeichenkette B$ eingelesene Wert mit Hilfe der CVS–Funktion als einfachgenauer numerischer Wert interpretiert.

Mit Hilfe der LOC–Funktion erhält man bei Direktzugriffsdateien die aktuelle Satznummer. Das ist die Satznummer, die bei der letzten ausgeführten GET– oder PUT–Anweisung verwendet wurde. Z.B. beendet die Anweisung

```
IF LOC(1)>50 THEN END
```

das Programm, wenn die aktuelle Satznummer für die Datei #1 größer als 50 ist.

Datenverarbeitung mit Direktzugriffsdateien Das folgende Programm illustriert für einen konkreten Anwendungsfall die Datenverarbeitung mit einer Direktzugriffsdatei.

Programm 6: Inventur

Im Gegensatz zu den vorhergehenden Beispielen werden bei diesem Programm wieder Zeilennummern verwendet, um zu zeigen, daß Amiga Basic auch solche "herkömmlichen" Basic–Programme interpretieren kann.

Programm 6: Lagerhaltung

```
120 OPEN "R",#1,"LAGER.DAT",39
125 FIELD #1,1 AS F$,30 AS D$,2 AS Q$,2 AS R$,4 AS P$
130 CLS:PRINT:PRINT "Programmfunktionen:":PRINT
135 PRINT "1: Datei einrichten"
140 PRINT "2: Neuen Eintrag erstellen"
150 PRINT "3: Lagerbestand für ein Teil anzeigen"
160 PRINT "4: Lagerbestand aufstocken"
170 PRINT "5: Entnahme aus dem Lager"
180 PRINT "6: Lagerbestaende unter Neubestellschwelle
        anzeigen"
190 PRINT "7: Programmende
220 PRINT:PRINT:INPUT "Welche Funktion";FUNK
225 IF (FUNK<1) OR (FUNK>7) THEN PRINT "Falsche
    Funktion":BEEP:FOR l=1 TO 500:NEXT:GOTO 130
230 ON FUNK GOSUB 900,250,390,480,560,680,970
240 GOTO 130
250 REM Neuen Eintrag erstellen
260 GOSUB 840
```

Programm 6: Lagerhaltung (Fortsetzung)

```
270 IF ASC(F$)<>255 THEN INPUT "Ersetzen (j/n)";A$:
    IF A$
280 LSET F$ = CHR$(0)
290 INPUT "Bezeichnung des Teils"; BEZ$
300 LSET D$=BEZ$
310 INPUT "Stückzahl";M%
320 LSET Q$=MKI$(M%)
330 INPUT "Neubestellschwelle";NS%
340 LSET R$=MKI$(NS%)
350 INPUT "Stueckpreis";STP
360 LSET P$=MKS$(STP)
370 PUT #1,TEIL%
380 RETURN
390 REM Datensatz anzeigen
400 GOSUB 840
410 IF ASC(F$)=255 THEN PRINT "Kein Eintrag":RETURN
420 PRINT USING "Teilenummer ###";TEIL%
430 PRINT D$
440 PRINT USING "Lagerbestand #####";CVI(Q$)
450 PRINT USING "Nachbestellschwelle #####";CVI(R$)
460 PRINT USING "Stueckpreis DM ##.##";CVS(P$)
465 FOR I=1 TO 3000:NEXT
470 RETURN
480 REM Lagerbestand aufstocken
490 GOSUB 840
500 IF ASC(F$)=255 THEN PRINT "Kein Eintrag":RETURN
510 PRINT D$:INPUT "Menge Zugang";A%
520 Q%=CVI(Q$)+A%
530 LSET Q$=MKI$(Q%)
540 PUT #1,TEIL%
550 RETURN
560 REM Entnahme aus dem Lager
570 GOSUB 840
580 IF ASC(F$)=255 THEN PRINT "Kein Eintrag":RETURN
590 PRINT D$
600 INPUT "Menge Entnahme";S%
610 Q%=CVI(Q$)
620 IF (Q%-S%)<0 THEN PRINT "Nur noch"Q%"Teile auf
    Lager":GOTO 600
630 Q%=Q%-S%
640 IF Q%<=CVI(R$) THEN PRINT "Bestand jetzt"Q%
    "Nachbestellschwelle ist"CVI(R$)
650 LSET Q$=MKI$(Q%)
660 PUT #1,TEIL%
665 FOR I=1 TO 3000:NEXT
670 RETURN
680 REM Teile unter Neubestellschwelle anzeigen
690 FOR I=1 TO 100
710 GET #1,I
```

Programm 6: Lagerhaltung (Fortsetzung)

```
720 IF CVI(Q$)<CVI(R$) THEN PRINT D$"Menge"CVI(Q$)
    TAB(50)"Neubestellschwelle"CVI(R$)
730 NEXT
740 RETURN
840 INPUT "Teilenummer";TEIL%
850 IF (TEIL%<1) OR (TEIL%>100) THEN PRINT "Falsche
    Teilenummer": GOTO 840 ELSE GET #1,TEIL%: RETURN
900 REM Datei einrichten
910 INPUT "Sind Sie sicher(j/n)";B$: IF B$<>"j" AND
    B$<>"J" THEN RETURN
920 LSET F$=CHR$(255)
930 FOR I=1 TO 100
940 PUT #1,I
950 NEXT
960 RETURN
970 CLOSE: END
```

Hauptprogramm; Zeilen 120–240

Das Hauptprogramm eröffnet die Datei, zeigt ein Menü mit den vorhandenen Programmfunktionen auf dem Bildschirm an und verzweigt in die vom Anwender gewählte Subroutine.

Zeile 120 Die Datei LAGER.DAT wird für wahlfreien Zugriff mit einer Satzlänge von 39 Bytes eröffnet und ihr wird der Puffer 1 zugeordnet.

Zeile 125 Die Felder des Dateipuffers werden nach Name und Länge definiert. F$ ist 1 Byte, D$ ist 30 Bytes, Q$ ist 2 Bytes, R$ ist 2 Bytes und P$ ist 4 Bytes lang.

Zeilen 130–190 Das Menü für die Auswahl einer Programmfunktion wird angezeigt.

Zeilen 220–225 Vom Anwender wird eine Eingabe gefordert, die auf Gültigkeit geprüft wird. Ist sie ungültig, wird eine Fehlerverzweigung angezeigt und der Anwender wird an das Menü zurückverwiesen.

Zeile 230 Verzweigt abhängig vom eingegebenen Wert zur gewünschten Subroutine (1 verzweigt nach Zeile 900, 2 nach Zeile 250 usw.).

Zeile 240 Nach Rückkehr aus der gewählten Subroutine wird der Anwender wieder an das Menü verwiesen (Zeile 130).

Subroutine für die Erstellung neuer Einträge; Zeilen 250–380

Zunächst wird geprüft, ob der betreffende Satz nicht schon Daten enthält. Dann wird der Anwender zur Dateneingabe aufgefordert, und schließlich wird der neue Satz in die Datei geschrieben.

Zeile 260 Verzweigung zur Subroutine, die die Eingabe der gewünschten Teilenummer (und damit Satznummer) fordert und den so spezifizierten Satz von Disk liest.

Zeile 270 Das erste Zeichen des spezifizierten Satzes wird geprüft. Ist es nicht CHR$(255) (damit werden in der Subroutine ab Zeile 900 anfänglich alle Sätze der Datei eingerichtet), so muß der Anwender entscheiden, ob er den Satz, der ja dann nicht leer ist, ersetzen will. Falls nicht, wird zum Hauptprogramm zurückverzweigt.

Zeile 280 Das erste Byte im Satz wird mit CHR$(0) überschrieben, um zu zeigen, daß der Satz jetzt Daten enthält.

Zeilen 290–300 Vom Anwender wird die Eingabe der Bezeichnung für das neue Teil gefordert, die dann in die für D$ im FIELD–Puffer reservierten 30 Bytes übertragen wird.

Zeile 310 Vom Anwender wird die Eingabe der Stückzahl gefordert.

Zeile 320 Die Stückzahl wird in die dafür im FIELD–Puffer unter Q$ reservierten 2 Bytes übertragen. Da es sich um numerische Daten handelt, müssen sie vorher mit der MKI$–Funktion als Zeichenkette behandelt werden.

Zeilen 330–340 Vom Anwender wird die Eingabe der Nachbestellschwelle gefordert, die ebenfalls als Zeichenkette interpretiert und in die unter R$ reservierten 2 Bytes im FIELD–Puffer übertragen wird.

Zeilen 350–360 Vom Anwender wird die Eingabe eines Stückpreises gefordert, der hier mit Hilfe der MKI$–Funktion als Zeichenkette interpretiert wird, da es sich um einen einfachgenauen Wert mit Dezimalstellen, also keinen ganzzahligen Wert wie bei Q$ und R$, handelt. Der Stückpreis wird mit 4 Byte Länge unter P$ im FIELD–Puffer abgelegt.

Zeile 370 Der Pufferinhalt wird als Satz auf Disk geschrieben. Die Teilenummer dient dabei als Satznummer.

Zeile 380 Es wird zurück ins Hauptprogramm verzweigt.

Subroutine für die Anzeige eines Satzes; Zeilen 390–470

Diese Subroutine prüft zunächst, ob der spezifizierte Satz Daten enthält. Dann werden die Inhalte aller Felder formatiert angezeigt.

Zeile 400 Verzweigung zur Subroutine, die die Eingabe der gewünschten Teilenummer (und damit Satznummer) fordert und den so spezifizierten Satz von Disk liest.

Zeile 410 Es wird geprüft, ob der spezifizierte Satz Daten enthält.

Zeilen 420–460 Die Inhalte der Satzfelder werden mit Hilfe der PRINT USING–Anweisung formatiert auf dem Bildschirm angezeigt. Numerische Werte, die als Zeichenketten auf Disk gespeichert waren, werden dabei mit Hilfe der CVI– und CVS–Funktionen wieder als numerische Werte interpretiert.

Zeile 470 Es wird zurück ins Hauptprogramm verzweigt.

Subroutine für die Lageraufstockung; Zeilen 480–550

Zunächst wird wieder geprüft, ob der spezifizierte Satz Daten enthält. Dann wird vom Anwender die Eingabe der Anzahl gefordert, um die der Lagerbestand des spezifizierten Teils aufgestockt werden soll. Um diese Anzahl wird dann der Bestand erhöht.

Zeile 490 Verzweigung zur Subroutine, die die Eingabe der gewünschten Teilenummer (und damit Satznummer) fordert und den so spezifizierten Satz von Disk liest.

Zeile 500 Es wird geprüft, ob der spezifizierte Satz Daten enthält.

Zeile 510 Zeigt die Teil–Bezeichnung auf dem Schirm an und fordert die Eingabe der Anzahl, um die der Lagerbestand für das betreffende Teil aufgestockt werden soll.

Zeile 520 Der Lagerbestand wird durch Addition erhöht. Dazu muß vorher der als Zeichenkette gespeicherte alte Bestand (Q$) als Ganzzahlwert mit Hilfe der CVI–Funktion interpretiert werden.

Zeile 530 Der neue Lagerbestand (Q%) wird wieder als Zeichenkette interpretiert und im Feld Q$ des FIELD–Puffers gespeichert. Der dort vorher gespeicherte alte Bestand wird dabei überschrieben.

Zeile 540 Der Pufferinhalt wird als Satz zurück auf die Disk geschrieben. Der alte Satz wird dabei überschrieben.

Zeile 550 Es wird zurück ins Hauptprogramm verzweigt.

Subroutine für die Entnahme aus dem Lager; Zeilen 560–670

Zunächst wird wieder geprüft, ob der spezifizierte Satz Daten enthält. Dann wird vom Anwender die Eingabe der Anzahl der spezifizierten Teile gefordert, die vom Lager entnommen werden sollen. Dabei wird geprüft, ob der Lagerbestand dafür ausreicht. Nach der Bestandsverminderung wird außerdem geprüft, ob die Nachbestellschwelle unterschritten ist. Der neue Lagerbestand für das spezifizierte Teil wird zurück auf Disk geschrieben.

Zeile 570 Verzweigung zur Subroutine, die die Eingabe der gewünschten Teilenummer (und damit Satznummer) fordert und den so spezifizierten Satz von Disk liest.

Zeile 580 Es wird geprüft, ob der spezifizierte Satz Daten enthält.

Zeile 590–600 Die Teil–Bezeichnung wird auf dem Schirm angezeigt, und es wird die Anzahl der spezifizierten Teile gefordert, die vom Lager entnommen werden sollen.

Zeile 610 Der alte Bestand, der als Zeichenkette Q$ gespeichert war, wird als Ganzzahlwert interpretiert und der Variablen Q% zugewiesen.

Zeile 620 Es wird geprüft, ob der Lagerbestand des spezifizierten Teiles die gewünschte Entnahme erlaubt. Falls nicht, wird nach einer Fehlermeldung zur Mengeneingabe (Zeile 600) zurückverzweigt.

Zeile 630 Der Lagerbestand wird durch Subtraktion vermindert.

Zeile 640 Es wird geprüft, ob jetzt die Nachbestellschwelle erreicht oder unterschritten ist. Falls ja, wird der Anwender durch eine Meldung darauf hingewiesen.

Zeile 650 Der neue Lagerbestand (Q%) wird als Zeichenkette interpretiert und wieder bei Q$ im FIELD–Puffer gespeichert.

Zeile 660 Der Pufferinhalt wird als Satz zurück auf Disk geschrieben. Der alte Satz wird dabei überschrieben.

Zeile 670 Es wird zurück ins Hauptprogramm verzweigt.

Subroutine zur Anzeige aller Teile, deren Anzahl unter der Nachbestellschwelle liegt; Zeilen 680–740

Diese Subroutine liest in einer Schleife jeden Satz und vergleicht den Bestand mit der für dieses Teil festgelegten Nachbestellschwelle. Liegt der Bestand unter der Nachbestellschwelle, wird eine entsprechende Meldung angezeigt.

Zeile 690 Es wird eine Programmschleife für alle Teilenummern eingerichtet, die die Programmzeilen 690 bis 730 umfaßt.

Zeile 710 Ein Satz wird von Disk in den Puffer gelesen.

Zeile 720 Bestand und Nachbestellschwelle werden für das betreffende Teil verglichen. Liegt der Bestand unter der Nachbestellschwelle, wird eine Meldung angezeigt.

Zeile 730 Der Schleifenzähler wird erhöht und die Schleife wird wiederholt.

Zeile 740 Nach hundert Schleifendurchläufen wird ins Hauptprogramm zurückverzweigt.

Subroutine zum Einlesen des gewünschten Satzes; Zeilen 840–850

Diese Subroutine dient zum Einlesen individueller Sätze aus der Datei. Sie wird von jeder anderen Subroutine angesprungen, in der die Daten des gewünschten Satzes verändert werden.

Zeile 840 Vom Anwender wird die Eingabe der gewünschten Teilenummer (und damit Satznummer) gefordert.

Zeile 850 Wenn eine ungültige Teilenummer (kleiner als 1 oder größer als 100) eingegeben wurde, wird eine Meldung angezeigt, und die Eingabe muß wiederholt werden (Zeile 840). Andernfalls wird der spezifizierte Satz von Disk gelesen und zurückverzweigt.

Subroutine zum Einrichten der Datei; Zeilen 900–960

Diese Subroutine richtet die Lagerdatei ein, indem als erstes Zeichen jedes Satzes CHR$(255) eingetragen wird. Damit wird der Satz als leer gekennzeichnet.

Zeile 910 Der Anwender muß bestätigen, daß er die Datei neu einrichten will. Hier kann er die Subroutine noch verlassen, ohne die Datei zu verändern.

Zeile 920 Die Kennung für leere Sätze (CHR$(255)) wird in die erste Stelle des FIELD–Puffers eingetragen.

Zeilen 930–950 In einer Schleife werden alle Sätze mit dem Inhalt des Puffers beschrieben.

Zeile 960 Es wird ins Hauptprogramm zurückverzweigt.

Programmende

Zeile 970 Die Datei wird geschlossen und die Programmausführung wird beendet.

5.5 Datenaustausch zwischen Amiga Basic und Textprogrammen

Wie Sie vielleicht wissen, erzeugen Textprogramme gewöhnlich Dateien, in denen nicht nur sichtbare Zeichen, sondern auch Spezialzeichen für die Textformatsteuerung enthalten sind, und die nicht sichtbar sind. Solche Zeichen können, wenn mit dem Textprogramm ein Amiga Basic–Programm geschrieben wurde, das Programm für den Interpreter unlesbar machen.

Nun verfügen die meisten Textprogramme über einen wählbaren Modus, in dem reine Textdateien ohne jede Steuerzeichen erstellt aber auch gelesen werden können.

Wenn Sie andererseits ein Programm mit Amiga Basic erstellen und es später für eine Veröffentlichung mit einem Textprogramm weiter bearbeiten wollen, speichern Sie es mit dem SAVE–Befehl unter Verwendung der **A**–Option. Dann wird das Programm im ASCII–Format, also im Klartext, gespeichert, das von den meisten Textprogrammen gelesen werden kann.

6. Besonderheiten von Amiga Basic

Amiga Basic verfügt über einige Programmierungsbesonderheiten wie Unterprogramme, Unterbrechungsreaktionsfähigkeit und Speicherverwaltung. Diese leistungsfähigen Eigenschaften, die vom Anfänger nicht unbedingt verstanden werden müssen, erhöhen die Flexibilität von Amiga Basic erheblich. Sie sind speziell für den fortgeschrittenen Programmierer hilfreich, der Programme für professionelle Anwendungen entwickelt.

Unterprogramme sind Module, die den herkömmlichen Subroutinen stark ähneln, gegenüber diesen jedoch erhebliche Vorteile aufweisen. Programmierer, die ihre Programme aus Moduln zusammensetzen, die auch in anderen Programmen Anwendung finden sollen, werden sich dieser Technik gerne bedienen.

Die Unterbrechungsreaktionsfähigkeit erlaubt einem Programm die Übergabe der Programmsteuerung an eine bestimmte Zeile, sobald ein bestimmtes Ereignis eintritt, wie z.B. Zeitablauf, Maus–Aktivitäten, Programmunterbrechung durch den Anwender oder Menü–Auswahl.

Mit Hilfe des CLEAR–Befehls und der FRE–Funktion kann der Anwender von seinem Programm aus den Amiga Basic–Arbeitsspeicher in gewissem Sinne verwalten und z.B. auch große Programme, bei denen es sonst Speicherkapazitätsschwierigkeiten geben kann, ablaufen lassen.

Die Amiga–Bibliotheksroutinen sind Maschinensprache–Routinen, die beim Starten des Rechners automatisch in den Speicher geladen werden. Um jedoch eine spezielle Routine von einem Amiga Basic–Programm aus nutzen zu können, muß die entsprechende Bibliothek zunächst geöffnet werden. Nach dem Aufruf der Routine muß die Bibliothek wieder geschlossen werden.

6.1 Unterprogramme

Wie normale Subroutinen bestehen auch Unterprogramme aus einer Reihe von Programmzeilen. Es gibt jedoch drei wesentliche Vorteile gegenüber den Subroutinen:

1. Unterprogramme verwenden lokale Variablen, also Variablen, die von den Variablen des Hauptprogramms isoliert sind. Wenn also ein Programmierer irrtümlich in einem Unterprogramm einen Variablennamen verwendet, der

bereits schon im Hauptprogramm existiert, so behalten beide Variablen ihre jeweiligen Werte. Der Wert einer Variablen im Unterprogramm kann also nicht vom Hauptprogramm aus verändert werden.

2. Auch der zweite Vorteil bezieht sich auf die lokalen Variablen. Es geschieht häufig, daß ein Programmierer in verschiedenen Programmen eine bestimmte Programmroutine immer wieder neu schreibt und sie an die Variablen des jeweiligen Programms anpaßt. Da Amiga Basic eben lokale Variablen unterstützt, empfiehlt es sich hier, einen Satz von Unterprogrammen mit festen lokalen Variablen zu schreiben, die in beliebigen Programmen ohne Änderung eingesetzt werden können, indem sie einfach mit dem MERGE–Befehl angefügt werden.

3. Unterprogramme können nicht ungewollt ausgeführt werden, Subroutinen dagegen leicht, wenn vor dem Beginn keine END– oder eine ähnliche Anweisung steht, die Programmausführung also einfach mit der ersten Zeile der Subroutine fortgesetzt wird. Unterprogramme werden dagegen ohne CALL–Anweisung niemals ausgeführt.

Der Aufruf von Unterprogrammen

Unterprogramme werden mit der CALL–Anweisung (s. dort in Kapitel 9) sowie wahlweise einer Liste von zu übergebenden Argumenten aufgerufen.

Im folgenden werden "Argumente" und "formale Parameter" unterschieden. Argumente sind Programmvariablen, die mit der CALL–Anweisung an das Unterprogramm übergeben werden. Z.B.

```
CALL STEUER(ZWISUM,MWST,GESAMTSUM())
```

Bei diesem Beispiel sind die beiden einfachen Variablen ZWISUM und MWST sowie die Feldvariable GESAMTSUM die Argumente.

Formale Parameter dagegen beziehen sich auf die parallel vom Unterprogramm verwendeten Werte. Wenn z.B. das STEUER–Unterprogramm im obigen Beispiel aufgerufen wird und als erste Zeile

```
SUB STEUER(BETRAG,STEUERSTUFE,SUM(1)) STATIC
```

enthält, so sind hier die Variablen BETRAG,STEUERSTUFE und SUM() formale Parameter. Diesen Parametern entsprechen (und werden übergeben) die als

Argumente verwendeten Hauptprogrammvariablen ZWISUM, MWST und GESAMTSUM.

Die vom Hauptprogramm zum Unterprogramm und zurück transferierten Parameter werden als "durch Referenz übertragen" bezeichnet. Das bedeutet: Wenn der formale Parameter vom Unterprogramm modifiziert wird, so ändert sich der Wert des Argumentes entsprechend.

Dies kann den Wert von Variablen beeinflussen. Z.B.:

```
CALL ADDIERE(A,B,C)
    .
    .
    .
SUB ADDIERE(X,Y,Z) STATIC
    Z=X+Y
    X=X+12
    Y=Y+94
END SUB
```

Wenn z.B. die Werte der Variablen A und B beim Aufruf des Unterprogramms ADDIERE durch die CALL–Anweisung A=2 und B=3 sind, so haben sich diese Werte nach der Rückkehr ins Hauptprogramm geändert, da die Variable A an die Variable X und die Variable B an die Variable Y gekoppelt sind. Wenn sich nun der Wert von X im Unterprogramm ändert, wird der Wert von A ebenfalls verändert. Im obigen Beispiel wird A um 12 vergrößert, weil X=X+12. X ist also die "Alias"–Variable zu A.

Wenn Sie dagegen nicht wollen, daß die Werte von Hauptprogrammvariablen sich durch Unterprogrammoperationen ändern, setzen Sie die Variablen in der Argumentliste in Klammern. Dies erhält den Wert der Variablen, den sie beim Aufruf des Unterprogramms hatten, unabhängig von jeglichen Operationen im Unterprogramm. Also z.B.:

```
CALL ADDIERE((A),(B),SUM)
```

Die Klammern um die ersten beiden Variablen erheben diese in die Kategorie von Ausdrücken, und Ausdrücke können in Unterprogrammen nicht verändert werden.

Wenn Sie dagegen ganze Ausdrücke an ein Unterprogramm übergeben wollen, benötigen Sie keine Klammern. Z.B.:

```
CALL ADDIERE(1+2,3*A,SUM)
```

Beachten Sie aber, daß der Typ der Argumente mit dem Typ der formalen Parameter übereinstimmen muß. Andernfalls wird eine "Type mismatch"-Fehlermeldung (fehlende Typübereinstimmung) angezeigt. Die Anweisungsfolge

```
CALL UPRO(1)
SUB UPRO(X) STATIC
```

ergibt eine Fehlermeldung, weil die ganze Zahl 1 der Gleitpunktvariablen einfacher Genauigkeit X übergeben wird. Diesen Fehler kann man mit

```
CALL UPRO(1.0)
SUB UPRO(X) STATIC
```

vermeiden.

Unterprogramm–Begrenzung: Die SUB– und END SUB–Anweisungen

Unterprogramme werden mit der SUB–Anweisung eingeleitet und mit der END SUB–Anweisung beendet. Zusätzlich kann aus einem Unterprogramm mit der EXIT SUB–Anweisung ausgesprungen werden, ehe die END SUB–Anweisung erreicht wird. Die END SUB– oder EXIT SUB–Anweisungen übergeben die Programmsteuerung zurück an den aufrufenden Programmteil. Es gilt folgende allgemeine Syntax:

```
SUB Name [(Liste form. Param.)] STATIC
[SHARED Variablenliste]
    .
    .
    .
END SUB
```

Für ***Name*** kann jeder beliebige Name bis zu einer Länge von 40 Zeichen nach den Amiga Basic–Namensregeln (s. Kapitel 8.3 und 8.5) gesetzt werden. Dieser Name darf dann jedoch nicht in irgendeiner anderen SUB–Anweisung verwendet werden.

In ***Liste form. Param*** dürfen zwei Typen von Einträgen stehen: einfache Variablen und Feldvariablen. Bei der Verwendung von Feldvariablen berücksichtigen Sie bitte die im nächsten Abschnitt angegebenen Regeln. Jeder Parameter wird vom nächsten durch ein Komma getrennt. Die Anzahl der Parameter wird nur durch die Maximallänge einer logischen Amiga Basic–Programmzeile begrenzt.

Das Schlüsselwort **STATIC** besagt, daß alle Variablen im Unterprogramm ihre Werte zwischen zwei Unterprogrammaufrufen behalten. Die Werte von mit STATIC deklarierten Variablen können nicht vom Hauptprogramm verändert werden. Außerdem bedeutet die STATIC–Option, daß das betreffende Unterprogramm nicht rekursiv ist, d.h. sich nicht selbst noch ein anderes Unterprogramm aufrufen darf, das dann seinerseits das aufrufende Unterprogramm wieder aufruft.

Variablen, die mit der SHARED–Anweisung (s. dort in Kapitel 9) deklariert werden, dürfen auch außerhalb des Unterprogramms verändert werden. Alle Variablen, für die das gelten soll, müssen in der auf die SHARED–Anweisung folgenden ***Variablenliste*** deklariert werden. Grundsätzlich werden zunächst alle in einem Unterprogramm verwendeten einfachen oder Feldvariablen als lokal betrachtet, es sei denn, sie werden explizit mit einer SHARED–Anweisung umdefiniert.

Die Anweisungen, die den Kern des Unterprogramms bilden, werden grundsätzlich von den SUB– und END SUB–Anweisungen am Anfang und Ende begrenzt.

In Unterprogrammen dürfen alle Amiga Basic–Befehle, –Anweisungen, –Funktionen und Systemvariablen verwendet werden, mit Ausnahme von:

- anwenderspezifischen Funktionsdefinitionen.
- Unterprogrammstrukturen. D.H. Unterprogramme können im Gegensatz zu Subroutinen **nicht** geschachtelt werden.
- COMMON–Anweisungen.
- CLEAR–Anweisungen.

Verwendung der SHARED– und STATIC–Anweisungen in Unterprogrammen

Variablen und Felder, auf die in Unterprogrammen Bezug genommen wird, oder die in diesen deklariert werden, gelten zunächst als lokal in diesem Unterprogramm. Amiga Basic erlaubt jedoch einerseits die Deklaration von globalen Variablen innerhalb eines Programms und andererseits den Erhalt von Werten über Unterprogrammaufrufe hinweg.

SHARED–Variablen Innerhalb eines Unterprogramms können Variablen des aufrufenden Hauptprogramms mit Hilfe der SHARED–Anweisung verwendet werden. In diesem Fall sind nur Variablen in dem Unterprogramm betroffen, in dem auch die SHARED–Anweisung verwendet wird.

Ein Beispiel:

```
LET A=1:LET B=5:LET C=10
DIM P(100),Q(100)
    .
    .
    .
SUB AMIGA STATIC
    SHARED A,B,P(),Q()
    .
    .
    .
END SUB
```

In diesem Beispiel werden alle Variablen des Hauptprogramms mit Ausnahme der Variablen C als mit dem Unterprogramm AMIGA gemeinsame Variablen deklariert.

STATIC–Variablen Wie bereits erwähnt, werden Variablen und Felder, die in einem Unterprogramm deklariert werden oder auf die dort Bezug genommen wird, für dieses Unterprogramm als lokal angenommen. Sie können nicht durch das Hauptprogramm oder andere außerhalb liegende Programmteile verändert werden. Ohne Zuweisungen haben diese Variablen den Wert 0 bzw. sind eine Zeichenkette mit der Länge 0. Wird das Unterprogamm verlassen und dann neu aufgerufen, so haben die Variablen die Werte vom letzten vorausgegangenen Aussprung.

Das Schlüsselwort STATIC wird bei allen Unterprogrammdefinitionen in Amiga Basic benötigt.

Parameterübergabe bei Unterprogrammen

Einfache Variablen und Feldelemente Wenn eine einfache Variable, ein Feldelement oder ein ganzes Feld an ein Amiga Basic–Unterprogramm übergeben wird, so gilt diese Übergabe als durch Referenz übertragen. Im folgenden Beispiel wird gezeigt, wie dies funktioniert:

```
DIM B(15)
A=4
CALL QUADRIERE(A,B(3))
PRINT A,B(3)
END

SUB QUADRIERE(X,Y) STATIC
    X=X+1
    Y=X*X
END SUB
```

Dieses Beispiel erzeugt als Ergebnis die Ausgabe 5 und 25. Jede Bezugnahme auf die Variable Y im Unterprogramm QUADRIERE resultiert in einer Bezugnahme auf das dritte Element des Feldes B, und jedesmal, wenn das Unterprogramm QUADRIERE die Variable X benutzt, wird auch A verändert.

Deklaration von Feld–Variablen als Parameter Einfachen Variablen in einer Parameterliste dürfen beliebige gültige Amiga Basic–Namen gegeben werden. Wenn jedoch ein ganzes Feld übergeben werden soll, so muß dies folgendermaßen deklariert werden:

```
Feldname([Dimensionen])
```

Feldname ist dabei ein beliebiger gültiger Amiga Basic–Name. Für die optional angebbare Zahl von ***Dimensionen*** wird eine Ganzzahlkonstante deklariert. Beachten Sie, daß die aktuellen Dimensionen hier **nicht** angegeben werden.

So sind z.B. in dem Unterprogramm

```
CALL MATADD2(X%,Y%,P(),Q(),R())
END
.
.
.
SUB MATADD(N%,M%,A(2),B(2),C(3)) STATIC
    .
    .
    .
END SUB
```

die Variablen N% und M% Ganzzahlvariablen. A und B sind als 2–dimensionale Felder und C als 3–dimensionales Feld deklariert. In der entsprechenden Argumentliste im Hauptprogramm werden die Klammern nur zur Kennzeichnung von Feldern benötigt.

Funktionen zur Bestimmung von Feldbegrenzungen Die Unter– und Obergrenze von Felddimensionen kann mit Hilfe der Funktionen LBOUND und UBOUND (s. dort in Kapitel 9) bestimmt werden.

LBOUND liefert die Untergrenze einer Dimension, also je nach mit der OPTION BASE–Anweisung (s. dort in Kapitel 9) gesetztem Wert 0 oder 1. Die voreingestellte Untergrenze für jede Dimension ist 0. UBOUND liefert die Obergrenze für die angegebene Dimension.

Für beide Funktionen existiert eine allgemeine und eine verkürzte Syntax, letztere für eindimensionale Felder:

LBOUND(*Feld*)	für 1–dimensionale Felder
LBOUND(*Feld,Dim*)	für n–dimensionale Felder
UBOUND(*Feld*)	für 1–dimensionale Felder
UBOUND(*Feld,Dim*)	für n–dimensionale Felder

Dabei müssen ***Feld*** ein gültiger Amiga Basic–Variablenname und ***Dim*** eine ganze Zahl zwischen 1 und der Maximalzahl der für das betreffende Feld vereinbarten Dimensionen sein.

LBOUND und UBOUND sind hilfreiche Funktionen bei der Bestimmung der Größe eines Feldes, das an ein Unterprogramm übergeben werden soll.

Ausdrücke als Argumente Wie bereits erwähnt, können auch Ausdrücke an Amiga Basic–Unterprogramme übergeben werden. Es dürfen hier beliebige gültige Amiga Basic–Ausdrücke mit Ausnahme von Referenzen auf einfache oder Feldvariablen verwendet werden. Findet der Interpreter in der Argumentliste einer CALL–Anweisung einen Ausdruck, so wird dieser einer temporären Variablen desselben Typs zugewiesen, die durch Referenz an das Unterprogramm übertragen wird. Dies ist praktisch derselbe Vorgang wie die Übergabe durch einen Funktionsaufruf über den Wert, wobei der Wert selbst übergeben wird.

Wird eine einfache Variable oder ein Feldelement in Klammern eingeschlossen, so wird dies als Ausdruck interpretiert und wie ein Ausdruck übergeben

(also Funktionsaufruf mit Wertübergabe. Wenn also im obigen Beispiel die CALL QUADRIERE–Anweisung in

```
CALL QUADRIERE ((A),B(3))
```

geändert würde, würde als Ergebnis 4 und 25 angezeigt. In diesem Fall würde nämlich (A) als Ausdruck mit seinem Wert übergeben und das Unterprogramm könnte den Wert von A nicht ändern.

6.2 Aufruf von Maschinensprache–Unterprogrammen

Genau wie Amiga Basic–Unterprogramme werden werden auch Maschinensprache–Unterprogramme mit der CALL–Anweisung aufgerufen. Die Binärdatei des Unterprogramms wird in den Speicher gelesen und dann ruft die CALL–Anweisung das Unterprogramm mit einer einfachen Variablen, die die Startadresse des Unterprogramms enthält, auf. Hierfür darf kein Feldelement verwendet werden.

Entsprechend den Unterprogrammaufruf–Konventionen der Programmiersprache C werden die Parameter mit ihrem Wert übergeben. Alle Parameter müssen vom Typ kurze oder lange Ganzzahl sein. Man kann aber auch die Adresse einer einfach– oder doppeltgenauen Gleitpunktvariablen mit Hilfe der VARPTR–Funktion oder die Adresse einer Zeichenkette mit Hilfe der SADD–Funktion (s. dort in Kapitel 9) übergeben. Beispielsweise werden mit dem Aufruf:

```
CALL MeinProg(VARPTR(ZZ),SADD(A$)
```

die Adressen der einfachgenauen Gleitpunktvariablen ZZ sowie der Zeichenkettenvariablen A$ übergeben.

Achtung: Es muß sichergestellt sein, daß die Maschinenroutine auf einer **geraden** Speicheradresse beginnt, sonst ist ein Systemabsturz unvermeidlich.

Achtung: Felder sollten mit den oben beschriebenen Prozeduren nicht an Maschinensprache–Unterprogramme übergeben werden. Statt dessen sollte lieber das Basis–Element des Feldes durch Referenz übertragen werden, wenn im Unterprogramm auf das ganze Feld zugegriffen wird, also z.B.

```
CALL XREF(VARPTR(A(0,0)))
```

Hier wird das erste Element eines zweidimensionalen Feldes an das Maschinensprache–Unterprogramm bei Startadresse XREF übergeben.

Im folgenden wird der Aufruf eines Maschinensprache–Unterprogramms aus einem Amiga Basic–Programm anhand eines Beispiels beschrieben. Zunächst das Maschinensprache–Unterprogramm, das eine Zeichenkette in Großbuchstaben wandelt:

```
                        SECTION  CODE
48E7  C080              MOVEM.L  A0/D0–D1,–SP      ;Reg. retten
202F  0010              MOVE.L   16(SP),D0         ;Länge holen
206F  0014              MOVE.L   20(SP),A0         ;Adr. 1. Stringbyte
4281                    CLR.L    D1                ;MSB von D1
                                                   löschen
6000  001C              BRA      Test              ;Schleife testen
                        ;
              Start:
1230  0000              MOVE.B   0(A0,D0),D1       ;nächstes
                                                   Stringbyte
0C01  0061              CMP.B    #'a',D1           ;wenn < a
6D00  0010              BLT      Test
0C01  007A              CMP.B    #'z',D1           ;oder > z
6E00  0008              BGT      Test              ;so lassen
                        ;
0230  00DF  0000        AND.B    #($FF–$20),0(A0,D0)  ;sonst wandeln
              Test:
51C8  FFE4              DBF      D0,Start          ;Zaehler dekrement.
4CDF  0103              MOVEM.L  (SP)+,A0/D0–D1    ;Register laden
4E75                    RTS                        ;Rueckkehr nach
                                                   Basic
```

Die von der Routine verwendeten Parameter werden zum Zeitpunkt des Aufrufes auf dem Stapel abgelegt; für das o.a. Unterprogramm CODE in der folgenden Ordnung:

	Offset
Zeichenketten–Adresse (adr&)	8 (SP) (SP = Stapelzeiger)
Zeichenketten–Länge (laeng&)	4 (SP)
Rückkehradresse	0 (SP)

Nach dem Ablegen der Register A0, D0 und D1 auf dem Stapel ist dessen Status folgendermaßen:

	Offset
Zeichenketten–Adresse (adr&)	20 (SP)
Zeichenketten–Länge (laeng&)	16 (SP)
Rückkehradresse	12 (SP)
A0	8 (SP)
D1	4 (SP)
D0	0 (SP)

Als nächstes wird das Amiga Basic–Programm gezeigt und beschrieben, das das o.a. Maschinensprache–Unterprogramm lädt, aufruft und die gewandelte Zeichenkette anzeigt:

```
WIDTH 70
     OPEN "CODE" FOR INPUT AS 1
     GrossCode$=INPUT$(LOF(1),1)
     CLOSE 1

INPUT "Text in Grop-/Kleinschreibung eingeben: ",st$
GOSUB Aufruf
END

Aufruf:
     Wandle&=SADD(GrossCode$)
     PRINT Wandle& ' Hier nur zur Kontrolle
     IF Wandle& AND 1 THEN GrossCode$=GrossCode$+" ":GOTO Aufruf
     adr&=SADD(st$)
     laeng&=LEN(st$)

     CALL Wandle&(laeng&,adr&)
     PRINT "Gewandelter Text:"
     PRINT st$
RETURN
```

Zunächst wird also die Binärdatei CODE in die Zeichenkette GrossCode$ geladen. Dies kann auf unterschiedliche Weise geschehen. Hier wurde eine Direktzugriffsdatei mit einem einzelnen Satz verwendet, in dem der gesamte Binärcode von 55 Bytes enthalten ist.

Der langen Ganzzahlvariablen **Wandle&** wird die Startadresse der binären Zeichenkette zugewiesen, die das Maschinensprache–Unterprogramm enthält. Amiga Basic weist dann eine temporäre Variable desselben Namens zu.

An das Maschinensprache–Unterprogramm werden zwei Argumente, nämlich **adr&**, die Adresse der zu umzucodierenden Zeichenkette sowie **laeng&**, deren Länge, übergeben. Sie werden im Anschluß an den Namen des Unterprogramms in Klammern angegeben. Dadurch werden sie beim Aufruf des Unterprogramms auf dem Stapel abgelegt (zuerst adr&, dann laeng&), von dem sie das Unterprogramm entnimmt.

Das Unterprogramm prüft alle Zeichen auf Kleinbuchstabe. Ist eines gefunden, so wird es durch den entsprechenden Großbuchstaben ersetzt. Alle anderen Zeichen werden nicht verändert. Am Ende der umzucodierenden Zeichenkette wird die Programmsteuerung wieder Amiga Basic übergeben. Im Basic–Programm wird schließlich die umcodierte Zeichenkette ausgegeben.

6.3 Aufruf von Bibliotheksroutinen

Bibliotheksroutinen sind spezielle Amiga–Programmdateien, die während der Laufzeit dynamisch mit Amiga Basic zusammengebunden werden. Um eine Bibliotheksroutine aufzurufen, wird die CALL–Anweisung in ähnlicher Weise verwendet wie zum Aufruf von selbstprogrammierten Maschinensprache–Routinen. Auch hier werden die Parameter mit ihrem Wert entsprechend den Aufrufkonventionen der Programmiersprache C übergeben. Um auf eine Bibliotheksroutine zugreifen zu können, müssen Sie zuerst die entsprechende Bibliothek öffnen.

Im folgenden wird ein Teil des Bibliotheksprogrammes, das auf der Extras–Diskette vorhanden ist, schrittweise beschrieben.

Eine Bibliothek öffnen

Für Ihre Amiga Basic–Anwendungen existieren eine ganze Reihe von Bibliotheken, von denen jede wiederum mehrere spezielle Routinen anthält. Jeder dieser Routinen ist eine besondere Informationsdatei zugeordnet, in der die erforderlichen Parameter und die zu verwendenen Prozessor–Register aufgeführt sind. Diese speziellen Dateien werden als **.fd**–Dateien bezeichnet.

Amiga Basic verwendet die Informationen in den fd.–Dateien etwas anders als der MC68000–Assembler oder der C–Compiler. Deshalb muß jede .fd–Datei in eine **.bmap**–Datei konvertiert werden, ehe eine Routine aus der Bibliothek von Amiga Basic aus verwendet werden kann. Das Hilfsprogramm **ConvertFD** für diese Konvertierung befindet sich in der **BasicDemos**–Schublade (–Unterverzeichnis) auf der **Extras**–Diskette.

Die Extras–Diskette enthält allerdings nur die .bmap–Dateien für die **dos.library**– und die **graphics.library**–Routinen. Wenn Ihre Anwendung Zugriff auf eine der **intuition.library**–Routinen erfordert, müssen Sie die .fd–Dateien für die Bibliothek intuition.library mit dem Hilfsprogramm **ConvertFD** in .bmap–Dateien konvertieren.

Im Anhang F finden Sie Details über das Format der .bmap–Dateien.

Sie öffnen eine Bibliothek mit der LIBRARY–Anweisung (s. dort in Kapitel 9). Unter der Voraussetzung, daß die eingelegte Diskette die entsprechenden .bmap–Dateien enthält, werden durch diese Anweisung alle die zur angesprochenen Bibliothek gehörenden Routinen für Ihr Amiga Basic–Programm verfügbar gemacht. Zur gleichen Zeit können bis zu fünf Bibliotheken geöffnet sein.

Eine Funktion aufrufen

Ist die Bibliothek erst einmal geöffnet, kann auf deren Routinen in ähnlicher Weise wie auf selbstgeschriebene Maschinensprache–Unterprogramme zugegriffen werden. Wenn von der aufgerufenen Routine jedoch ein Wert zurückerwartet wird, muß im Amiga Basic–Programm dessen Typ mit einer DECLARE FUNCTION–Anweisung (s. dort in Kapitel 9) festgelegt werden (z.B. durch das Zeichen & am Ende des Variablennamens bei einem langen Gleitpunkt–Wert).

Der folgende Teil des Bibliotheks–Demonstrationsprogrammes zeigt die Anwendung dieser Anweisungen:

```
DECLARE FUNCTION AskSoftStyle& LIBRARY
DECLARE FUNCTION OpenFont& LIBRARY
LIBRARY "graphics.library"
enable%=AskSoftStyle&(WINDOW(8))
Font "topaz.font",8,0,0
FOR i=0 TO 4
    SetStyle CINT(2↑i)
NEXT i
    .
    .
    .
SUB SetStyle(mask%) STATIC
    SHARED enable%
    SetSoftStyle WINDOW(8),mask%,enable%
    PRINT "SetSoftStyle (";mask%;")"
END SUB
```

Mit den DECLARE FUNCTION–Anweisungen wird Amiga Basic mitgeteilt, daß die beiden Funktionen **AskSoftStyle&** und **OpenFont&** aus der Bibliothek **graphics.library** Ganzzahlwerte liefern. Die Bibliothek selbst wird mit der LIBRARY–Anweisung in der dritten Programmzeile geöffnet.

Mit der darauf folgenden Anweisung wird die Routine bei **AskSoftStyle&** aufgerufen und deren zurückgegebener Wert der Variablen **enable%** zugewiesen. Beachten Sie hier, daß das Schlüsselwort CALL nicht erforderlich ist. Ausnahmen werden unten beschrieben. Wenn der Aufruf ausgeführt wird, richtet Amiga Basic eine temporäre Variable mit demselben Namen AskSoftStyle ein (die Kennung & wird dabei ignoriert).

Bei diesem Beispiel wird der zurückgegebene Wert zu einer kurzen Ganzzahl abgeschnitten. Er repräsentiert die acht Schriftarten–Bits des aktuellen Zeichensatzes. Die DECLARE FUNCTION–Anweisung könnte deshalb ebensogut eine kurze Ganzzahl–Variable verwenden:

```
DECLARE FUNCTION AskSoftStyle% LIBRARY
```

Ohne jede Deklaration würde Amiga Basic allerdings versuchen, eine Zuweisung in einfacher Genauigkeit auszuführen und das Ergebnis wäre falsch.

Die Funktion AskSoftStyle& benötig einen Parameter, nämlich die WINDOW-Funktion (s. dort in Kapitel 9), mit deren Hilfe Informationen über den aktuellen Zeichensatz gewonnen werden kann.

In dem Beispiel werden noch verschiedene andere Routinen aus der Bibliothek graphics.library verwendet, wobei für jede die Parameter, die von Amiga Basic übergeben werden, aufgelistet sind. Jede dieser Bibliotheksroutinen ist in dem AmigaROM–Handbuch beschrieben.

Explizite Verwendung des CALL–Schlüsselwortes

Die meisten Bibliotheksroutinen können, wie in dem o.a. Beispiel angegeben, ohne das Schlüsselwort CALL aufgerufen werden. Wenn jedoch der Aufruf auf eine ELSE– oder THEN–Klausel folgt, muß CALL angegeben werden, damit der Interpreter den Routinennamen nicht mit einer Sprungmarke verwechselt. Also z.B.:

```
IF pFont&<>0 THEN CALL CloseFont (pFont&)
```

6.4 Verarbeitung von Unterbrechungsereignissen

Die Unterbrechungsreaktion ist ein Hilfsmittel, mit dem ein Programm auf bestimmte Ereignisse reagieren und zu einem entsprechenden Programmteil verzweigen kann. Amiga Basic kann auf folgende Ereignisse reagieren:

- Zeitablauf (ON TIMER–Anweisung),
- Programmabbruch durch Anwender (ON BREAK–Anweisung),
- Die Wahl eines Menü–Punktes (ON MENU–Anweisung),
- Maus–Aktivität (ON MOUSE–Anweisung).

Bei aktivierter Unterbrechungsreaktionsfähigkeit prüft Amiga Basic vor der Ausführung einer jeden Anweisung, ob ein spezifiziertes Ereignis eingetreten ist.

In der Praxis erstellt der Programmierer eine Subroutine, um auf das Ereignis zu reagieren und aktiviert die Unterbrechungsreaktionsfähigkeit. Sobald dann das betreffende Ereignis eintritt, wird die Programmsteuerung automatisch an die Subroutine übergeben. Intern entspricht dies exakt einer GOSUB–Anweisung zu dieser Subroutine.

Als Abschluß der Subroutine wird durch eine RETURN–Anweisung die Programmsteuerung wieder an die Anweisung im Hauptprogramm zurückgegeben, die der letzten vor der Unterbrechung noch ausgeführten Anweisung folgt.

In diesem Abschnitt wird ein Überblick über die Programmiertechnik für Unterbrechungsreaktionen gegeben. Nähere Informationen über die oben erwähnten Anweisungen entnehmen Sie bitte den entsprechenden Abschnitten im Kapitel 9.

Die Unterbrechungsreaktionsfähigkeit wird durch folgende Anweisungen gesteuert:

Ereignis **ON**	schaltet Reaktionsfähigkeit ein,
Ereignis **OFF**	schaltet Reaktionsfähigkeit aus,
Ereignis **STOP**	schaltet Reaktionsfähigkeit vorübergehend aus.

Für ***Ereignis*** muß eines der folgenden Schlüsselwörter angegeben werden.

TIMER — Damit ist die Systemuhr des Amiga gemeint. Mit ihr kann der Ablauf einer vorgegebenen Anzahl von Sekunden überwacht werden.

MOUSE — Die Programmsteuerung kann vom Drücken einer der Auswahltaste der Maus abhängig gemacht werden.

MENU — Die Programmsteuerung kann von der Wahl eines Menüs aus der Menü–Leiste abhängig gemacht werden.

BREAK — Sobald der Anwender die Tastenkombination rechte Amiga-Taste – Punkt-Taste drückt, verzweigt das Programm zur angegebenen Subroutine. Diese Reaktionsmöglichkeit sollte jedoch mit Vorsicht verwendet werden. Wenn sie in einem zu testenden Programm verwendet wird, kann das Programm vor der END–Anweisung nicht anders als durch Rücksetzen des gesamten Systems beendet werden. Deshalb sollte zunächst die BREAK ON–Anweisung weggelassen werden und erst eingefügt werden, wenn das Programm vollständig ausgetestet ist.

COLLISION — Die Programmsteuerung wird von der Kollision eines mit der OBJECT.SHAPE–Anweisung (s. dort in Kapitel 9) erzeugten beweglichen, grafischen Objektes (Sprite oder Bob (s. Kapitel 7.4)) mit einem anderen Objekt oder dem Bildfensterrand abhängig gemacht.

ON. . .GOSUB–Anweisung

Mit dieser Anweisung wird die Anfangszeile der Verarbeitungs–Subroutine für das Unterbrechungsereignis festgelegt. Das Format ist

ON *Ereignis* **GOSUB** *Sprungmarke*

Wird für *Sprungmarke* eine 0 angegeben, so wird die Unterbrechungsreaktionsfähigkeit inaktiviert.

Aktivierung der Unterbechungsreaktionsfähigkeit Wenn die Unterbrechungsreaktionsfähigkeit für ein Ereignis aktiviert ist (ON) und eine von 0 verschiedene Sprungmarke in der ON. . .GOSUB–Anweisung angegeben

wurde, prüft der Amiga Baisc–Interpreter vor der Ausführung jeder Anweisung, ob das angegebene Ereignis eingetreten ist.

Solange die entsprechende *Ereignis* ON–Anweisung nicht gegeben wurde, findet jedoch keine Reaktion durch die entsprechnede ON *Ereignis* GOSUB-Anweisung statt.

Inaktivierung der Unterbrechungsreaktionsfähigkeit Wenn die Unterbrechungsreaktionsfähigkeit für ein Ereignis abgeschaltet ist (OFF), erfolgt keine Unterbrechung, und der Eintritt des Ereignisses wird auch nicht gespeichert.

Unterbrechungsreaktionsfähigkeit zeitweilig inaktivieren Wenn die Unterbrechungsreaktionsfähigkeit für ein Ereignis nur gestoppt ist (STOP), nicht aber abgeschaltet, erfolgt beim Eintritt des Ereignisses ebenfalls keine Unterbrechung, der Eintritt wird aber gespeichert. Sobald jetzt eine *Ereignis* ON–Anweisung gegeben wird, erfolgt sofort eine Unterbrechung.

Sobald eine Unterbrechungsreaktion stattgefunden hat, führt der Interpreter intern automatisch für das betreffende Ereignis eine *Ereignis* STOP–Anweisung durch, um Unterbrechungsschachtelung zu verhindern. Nach Rückkehr aus der Bearbeitungsroutine wird automatisch eine *Ereignis* ON–Anweisung ausgeführt, um die Unterbrechungsreaktionsfähigkeit für das entsprechende Ereignis wieder zu aktivieren, es sei denn, in der Unterbrechungsroutine selbst wurde explizit eine *Ereignis* OFF–Anweisung für das betreffende Ereignis gegeben.

Achtung: Hat eine Unterbrechungsreaktion stattgefunden, so bleibt die Reaktionsfähigkeit für dieses Ereignis solange abgeschaltet, bis eine RESUME–Anweisung (s. dort in Kapitel 9) ausgeführt wurde.

6.5 Speicherverwaltung

Mit Hilfe der CLEAR–Anweisung können Sie bei großen Programmen den Speicher für verschiedene Zwecke aufteilen.

Sie können die Größe von drei verschiedenen Speicherbereichen mit der CLEAR–Anweisung einstellen:

- den Stapel–Speicher;
- den Amiga Basic–Programmspeicher;
- den Amiga Basic–Systemspeicher.

Der Stapelspeicher

Der Stapelspeicher dient dem Amiga Basic–Interpreter zum Vermerken von Rückkehradressen bei Subroutinen– oder Unterprogrammaufrufen sowie zur Buchführung bei geschachtelten Schleifen oder Anwenderfunktionen.

Verringerung des Stapelspeicherplatzbedarfs Einige spezielle ROM–Routinen des Amiga haben, wenn aufgerufen, erheblichen Stapelspeicherbedarf. Außerdem steigt der Bedarf an Stapelspeicher, wenn ein Programm viele geschachtelte Schleifen enthält. Durch geschickte Programmiertechnik kann erheblich an Stapelspeicher gespart werden.

Der Amiga Basic–Programmspeicher

Im Programmspeicher ist der eigentliche Programmtext, also die einzelnen Programmzeilen, abgelegt. Ferner sind hier die Variablen des Programms sowie Zeichenketten gespeichert. Schließlich enthält der Programmspeicher die Datenpuffer für geöffnete Dateien.

Verringerung des Programmspeicherplatzbedarfs Der Puffer für eine sequentielle Datei ist mit 128 Bytes Länge voreingestellt. Durch Verkleinerung der Pufferlänge (s. OPEN–Anweisung in Kapitel 9) kann Speicher gespart werden. Dies geht jedoch auf Kosten der Verarbeitungsgeschwindigkeit. Einen weiteren Einfluß auf den Programmspeicherbedarf hat der Typ der verwendeten numerischen Variablen. Ganzzahlvariablen brauchen halb soviel Speicherplatz wie Variablen einfacher Genauigkeit, und diese halb soviel wie Variablen doppelter Genauigkeit. Außerdem benötigt die Verkettung mehrerer kleiner Programmsegmente mit der CHAIN–Anweisung (s. dort in Kapitel 9) weniger Speicher als ein großes Programm.

Der System–Speicher

Der Amiga Basic–Systemspeicher enthält die Puffer für die Informationen, die die SOUND– und WAVE–Anweisungen (s. dort in Kapitel 9) benötigen, wenn sie aufgerufen werden. In diesem Fall werden 1024 Bytes reserviert. Außerdem benötigen die Anweisungen WINDOW, SCREEN und LIBRARY Teile des Systemspeichers. Amiga Basic teilt sich den Systemspeicher mit anderen Programmen, die ggf. gleichzeitig ablaufen (Multitasking).

Verringerung des Systemspeicherplatzbedarfs Sobald der SOUND–/WAVE–Datenpuffer nicht mehr benötig wird, sollte er mit der ERASE–Anweisung wieder freigegeben werden. Dies verringert den Systemspeicherplatzbedarf immerhin um 1 KByte.

Speicherreservierung mit der CLEAR–Anweisung

Mit der CLEAR–Anweisung können Sie, wie schon erwähnt, für drei Speicherbreiche Platz reservieren. Die allgemeine Syntax der CLEAR–Anweisung ist folgende:

CLEAR [,[*Programmspeicher*][,*Stapelspeicher*]]

Das Argument *Programmspeicher* gibt an, wieviele Bytes für das eigentliche Amiga Basic–Programm reserviert werden sollen.

Das Argument *Stapelspeicher* gibt an, wieviel Bytes für den Amiga Basic-Stapelspeicher reserviert werden sollen.

Der verbleibende Speicherrest, also die Differenz aus dem Gesamtspeicher sowie der Summe aus Programm– und Stapelspeicher wird automatisch für den Systemspeicher reserviert. Mit dem CLEAR–Befehl können Sie also vom Programm aus den für die drei einstellbaren Speicherbereiche benötigten Platz festlegen. Mit Hilfe der FRE–Funktion (s. dort in Kapitel 9) können Sie jederzeit feststellen, wieviel freien Speicher Sie noch in jedem der drei Bereiche zur Verfügung haben (s. unten)

Anwendung der FRE–Funktion bei der Speicherverwaltung

Die FRE–Funktion hat folgende Syntax:

```
FRE(n)
```

mit drei verschiedenen Modi:

1. Bei $n=-1$ liefert FRE die Anzahl der noch verfügbaren Systemspeicherbytes.

2. Bei $n=-2$ liefert FRE die Anzahl der **noch nicht** benutzten Stapelspeicherbytes

3. Bei jeder anderen Zahl für *n* erhält man die Anzahl der noch verfügbaren Programmspeicherbytes.

Bei jedem Aufruf von FRE wird außerdem der Zeichenkettenspeicherbereich innerhalb des Programmspeichers "aufgeräumt" und komprimiert.

7. Erzeugen bewegter Bilder mit dem Objekt–Editor

Dieses Kapitel beschreibt den Objekt–Editor. Der Objekt–Editor ist ein Hilfsprogramm des Amiga Basic–Interpreters, mit dessen Hilfe grafische Objekte für Animations–Programme erzeugt werden können. Unter Animation versteht man die programmierte Bewegung von Abbildungen auf dem Bildschirm. Die Arbeitsweise des Objekt–Editors wird anhand eines schrittweise dargestellten Beispiels beschrieben.

7.1 Übersicht

Amiga Basic erschließt sich die Animationsmöglichkeiten des Amiga–Systems über Programm–Anweisungen und den Objekt–Editor. Die COLLISION– und OBJECT–Anweisungen (s. dort in Kapitel 9) manipulieren die grafischen Objekte im Ausgabe–Fenster. Mit Hilfe des Objekt–Editors werden diese **Objekte**, wie sie im weiteren Verlauf dieses Kapitels bezeichnet werden, definiert.

Mit dem Objekt–Editor können Sie:

- schnell Ovale, Rechtecke und Linien zeichnen, indem Sie den Maus–Zeiger, der hier die Funktion eines Zeichenstiftes hat, zwischen zwei Punkten auf der Zeichenfläche des Objekt–Editors bewegen. Die Zeichenfläche ist der Bereich des Ausgabe–Fensters, innerhalb dem Sie das Objekt entwerfen.

- frei auf der gesamten Zeichenfläche mit dem Zeichenstift des Objek–Editors zeichnen.

- Farben für die Umrandung des Objektes auswählen.

- das Innere des Objektes mit der gewählten Randfarbe ausmalen.

- beliebig in den Objekten löschen und ändern.

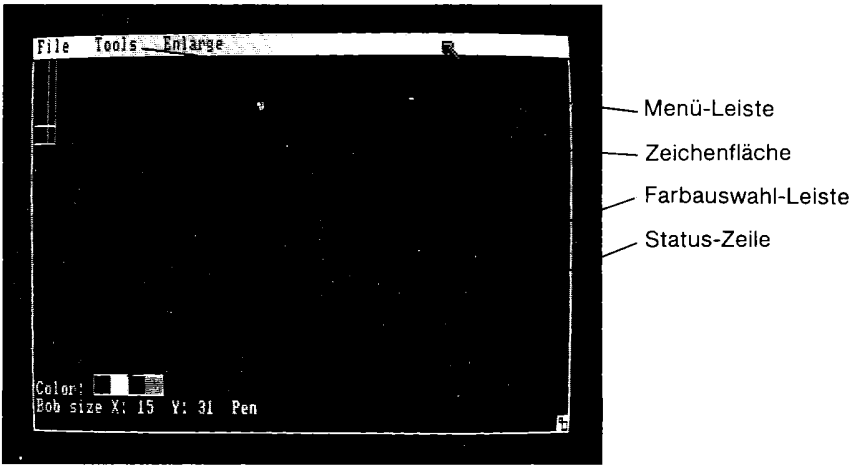
Nach der Erzeugung eines Objektes kann dieses in einer Datei mit wählbarem Namen gespeichert werden. Die Datei enthält dann die statischen Attribute für das Objekt, also Form, Größe und Farbe. Um einem solchen Objekt in einem

Programm Bewegung zu verleihen, wird die Objektdatei zunächst geöffnet, das Objekt wird als binäre Zeichenkette eingelesen und dann mit der OBJECT.SHAPE–Anweisung (s. dort in Kapitel 9) für das Programm definiert.

Achtung: Der Objekt–Editor weist den erzeugten Objekten Attribute zu, die die Berührung von Objekten untereinander oder von Objekten mit dem Fensterrand als Kollision auslegen. Diese Voreinstellung kann mit der OBJECT.HIT–Anweisung (s. dort in Kapitel 9) geändert werden, so daß sich Objekte z.B. auch durchdringen oder hintereinander vorbeibewegen können.

7.2 Das Fenster des Objekt–Editors

In diesem Abschnitt wird die Darstellung des Objekt–Editor–Fensters (s.u.) innerhalb des Bildschirmes beschrieben. Innerhalb dieses Fenster entwerfen Sie Ihre Objekte:



Das Fenster besteht aus einer Reihe von Elementen, die im folgenden näher erläutert werden:

Die Menü–Leiste

Im Fenster des Objekt–Editors können Sie zwischen folgenden Menüs wählen:

- File (Datei)
- Tools (Hilfsmittel)
- Enlarge (Vergrößern)

Das **File**–Menü enthält Funktionen zum Laden oder Speichern von Objekten von bzw. auf Diskette. Die Funktionen des **Tools**–Menü erleichtern das Entwerfen von Objekten. Mit den Funktionen des **Enlarge**–Menüs können Sie Ihre Objekte vergößert darstellen, um feine Details anzubringen oder zu ändern.

Die Zeichenfläche

Die Zeichenfläche befindet sich oben links im Fenster des Objekt–Editors. Innerhalb dieser Fläche zeichnen oder verändern Sie Ihr Objekt und können es bunt ausmalen.

Sie können die Größe der Zeichenfläche ganz einfach ändern, indem Sie das Größen–Symbol in der rechten unteren Fensterrahmen–Ecke mit gedrückter Auswahltaste der Maus ziehen.

Wenn Sie ein Sprite erzeugen (ein Sprite ist ein besonderer Typ von Objekt, der im weiteren Verlauf dieses Kapitels noch näher beschrieben wird), können Sie dessen Breite nicht über die angezeigte Breite hinaus (16 Bildpunkte, von 0 bis 15 gezählt), wohl aber dessen Höhe vergrößern.

Die Farbauswahl–Leiste

Mit Hilfe der Farbauswahl–Leiste links unten im Fenster der Objekt–Editors können Sie die Innen– und die Randfarben Ihres Objektes wählen oder verändern. Sie brauchen dazu nur auf die gewünschte Farbe zu zeigen und die Auswahltaste der Maus zu drücken. Das Wort **Color** (Farbe) links neben der Farbleiste wird dann in der gewählten Farbe angezeigt.

Die Anzahl der wählbaren Farben in der Farbauswahl–Leiste hängt von der Tiefe des Bildschirms ab, die mit der SCREEN–Anweisung (s. dort in Kapitel 9) eingestellt werden kann.

Um Objekte mit mehr als vier Farben zu erzeugen, müssen Sie das Objekt-Editor–Programm ändern. Dieses Programm ist in Amiga Basic geschrieben und ausführlich mit Kommentaren in der Programmliste versehen, so daß Änderungen für einen geübten Programmierer nicht schwierig sind. Das Programm, mit dem vielfarbige Objekte bewegt werden sollen, muß eine SCREEN–Anweisung mit entsprechender Bildschirm–Tiefeneinstellung enthalten.

Die Status–Zeile

in der unteren linken Ecke des Objekt–Editor–Fensters sehen Sie die x– und y–Koordinaten der Position innerhalb der Zeichenfläche, bei der die Auswahltaste der Maus das letzte Mal gedrückt wurde. Rechts daneben wird die aktuelle, zuletzt aus dem **Tools**–Menü gewählte Funktion (Pen = Zeichenstift, Oval = Oval, Rectangle = Rechteck, Paint = Ausmalen, Eraser = Löscher) angezeigt.

7.3 Die Menüs des Objekt–Editors

Sie können zwischen folgenden Punkten des **File**–Menüs wählen:

New	Löscht den Bildschirm und zeigt die Zeichenfläche in ihren voreingestellten Abmaßen.
Open	Fordert die Eingabe eines existierenden Namens für eine Datei, die vorher mit dem Objekt–Editor erstellt wurde. Die Eingabe wird durch Drücken der RETURN–Taste abgeschlossen.
Save	Speichert ein Objekt in einer Datei unter demselben Namen, unter dem diese geöffnet wurde. Falls Sie vorher die **New**-Funktion benutzt haben, müssen Sie einen Dateinamen eingeben. Die Eingabe wird in jedem Fall durch Drücken der RETURN–Taste abgeschlossen.

Save As	Fordert zur Speicherung eines Objektes den Dateinamen an. Drücken der RETURN–Taste beendet die Eingabe.
Quit	Beendet die Arbeit mit dem Objekt–Editor und bringt Sie zurück auf die Amiga Basic–Ebene.

Sie können zwischen folgenden Punkten des **Tools**–Menüs wählen:

Pen	Erlaubt das freie Zeichnen innerhalb der Zeichenfläche.
Line	Zeichnet eine Gerade zwischen zwei Punkten.
Oval	Zeichnet ein Oval.
Rectangle	Zeichnet ein Rechteck.
Erase	Löscht ein Bild innerhalb der Zeichenfläche.
Paint	Malt die Innenfläche eines Objektes mit der aktuell gewählten Farbe aus.

Sie können zwischen folgenden Punkten des **Enlarge**–Menüs wählen:

4x4	Vergrößert die Zeichenfläche um den Faktor 4. Dabei darf die Zeichenfläche nicht größer als maximal 100 Bildpunkte horizontal mal 31 Bildpunkte vertikal sein.
1x1	Zeigt die Zeichenfläche in ihrer ursprünglichen Größe an.

7.4 Eine Bemerkung über "Bobs" und "Sprites"

Beim Amiga wird zwischen zwei verschiedenen Typen von grafischen Objekten unterschieden. In der Terminologie des Amiga sind dies **Bobs** und **Sprites**. Dem Objekt–Editor müssen Sie vor der Erstellung oder Änderung eines Objekts mitteilen, ob es sich dabei um ein Bob oder ein Sprite handelt. Sie müssen deshalb vorher die Unterschiede zwischen beiden Objekt–Typen kennen. Ist dies bereits der Fall, so können Sie jetzt gleich zum nächsten Kapitel dieses Handbuches weitergehen.

In der nachfolgenden Tabelle werden die wesentlichen Unterschiede zwischen Bobs und Sprites einander gegenübergestellt:

Bobs	**Sprites**
lassen sich langsamer als Sprites bewegen.	lassen sich schneller als Bobs bewegen.
die Größe wird nur durch den verfügbaren Speicher begrenzt.	Die Breite ist auf 16 Bildpunkte begrenzt.
Es sind alle Farben erlaubt.	Es sind 3 Farben erlaubt.
Es können alle Bobs angezeigt werden.	Auf einer Zeile können zur selben Zeit nur vier Sprites mit verschiedenen Farben angezeigt werden.

Wenn Sie sich näher mit Bobs und Sprites beschäftigen wollen, finden Sie im AmigaROM–Benutzerhandbuch eine Fülle von weiteren Informationen.

7.5 Wie man Objekte erstellt

Den Objekt–Editor finden Sie auf der **Extras**–Diskette in der Ablage **BasicDemos** unter dem Namen **ObjEdit**. Der Objekt–Editor wird wie jedes andere Amiga Basic–Programm aktiviert (s.a. Kapitel 2).

Achtung: Wenn Sie mit einer 256–kBytes–Maschine arbeiten, müssen Sie das Piktogramm des Objekt–Editors aus dem Fenster der **BasicDemos** herausziehen. Anschließend schließen Sie alle Fenster und wählen dann das Objekt–Editor–Piktogramm. Wollen Sie den Objekt–Editor dagegen aus Amiga Basic heraus laden, verwenden Sie den Dateinamen **basicdemos/objedit**. Außerdem müssen Sie die Zeile mit der Anweisung

```
LIBRARY "graphics.librarary"
```

in

```
LIBRARY ":basicdemos/graphics.library"
```

ändern.

Für die Erstellung eines Objektes folgen Sie jetzt diesen Schritten:

1. Sobald der Objekt–Editor aktiviert ist, erhalten Sie folgende Anzeige:

   ```
   Enter 1 if you want to edit sprites
   Enter 0 if you want to edit bobs >
   ```

   (Geben Sie 1 ein, wenn sie Sprites erzeugen wollen
   Geben Sie 0 ein, wenn sie Bobs erzeugen wollen >)

   Geben Sie hier die gewünschte Kennung ein und drücken Sie die RETURN-Taste.

   Achtung: Versuchen Sie nicht, das Objekt–Editor–Fenster **hinter** andere Fenster, also in den Hintergrund, zu stellen.

2. Jetzt wird das Objekt–Editor–Fenster angezeigt. Wählen Sie aus dem **Files**-Menü **New**, wenn Sie ein neues Objekt zeichnen wollen, oder **Open**, wenn Sie ein existierendes Objekt ändern wollen.

3. Anschließend wählen Sie aus dem **Tools**–Menü, wie sie auf der Zeichenfläche arbeiten wollen. Wählen Sie die **Pen** wenn Sie frei zeichnen wollen, oder aber **Line**, **Oval** oder **Rectangle**, wenn Sie eine Gerade, ein Oval oder ein Rechteck zeichnen wollen. Mit **Erase** können Sie Teile des Objektes auch wieder löschen.

   Wenn Sie eine definierte Figur zwischen Punkten zeichnen wollen, brauchen Sie den Zeiger mit der Maus nur auf den Anfangspunkt zu stellen, die Auswahltaste der Maus niederzuhalten und dann die Maus zum Endpunkt zu bewegen. Anschließend geben Sie die Auswahltaste wieder frei. Der Zeichen– oder Löschvorgang wird außerdem automatisch beendet, sobald Sie mit dem Zeiger außerhalb der Zeichenfläche geraten und fortgesetzt, wenn Sie sich wieder innerhalb der Zeichenfläche befinden.

   Beachten Sie, daß, wenn Sie ein Oval zeichnen wollen, zunächst ein Rechteck auf der Zeichenfläche erscheint. Sobald Sie dann aber die Auswahltaste der Maus loslassen, wird dieses durch das Oval ersetzt.

4. Für Farbänderungen zeigen Sie einfach mit dem Zeiger auf die entsprechende Farbe in der Farbauswahl–Leiste links unten im Fenster und drücken dann die Auswahltaste der Maus. Der Objekt–Editor umrahmt dann jedes neu erzeugte Objekt auf der Zeichenfläche mit dieser Farbe.

5. Zum Ausmalen des Inneren eines Objektes wählen Sie ebenfalls die gewünschte Farbe aus der Farbauswahl–Leiste und anschließend **Paint** aus dem **Tools**–Menü. Zeigen sie dann mit dem Zeiger auf den Bereich, den Sie ausmalen wollen und drücken Sie die Auswahltaste der Maus.

   Die auszumalende Fläche sollte vollständig von einer Umrandung in derselben Farbe umgeben sein. Andernfalls würde die Farbe in den umgebenden Bereich "auslaufen".

6. Zur Veränderung der Zeichenflächen–Größe brauchen Sie nur mit dem Zeiger auf das Größen–Symbol in der rechten unteren Ecke der Zeichenfläche zu zeigen, die Auswahltaste der Maus zu drücken und die Maus dann zu bewegen.

   Amiga Basic behandelt die Zeichenfläche unabhängig von der Anzahl der auf ihr erzeugten Einzelbilder generell als ein Objekt. Unterschiedliche Objekte müssen deshalb auch auf getrennten Zeichenflächen entworfen und in getrennten Dateien gespeichert werden.

7. Ist das Objekt Ihren Vorstellungen entsprechend fertiggestellt, wählen Sie **Save As** aus dem File-Menü, wenn es sich um ein neues Objekt handelt, oder **Save,** wenn Sie ein bereits existierendes Objekt geändert haben, geben Sie ggf. den entsprechenden Dateinamen ein und drücken Sie abschließend die RETURN–Taste.

8. Die Elemente der Amiga Basic–Sprache

8.1 Der Zeichensatz

Der Amiga Basic–Zeichensatz besteht aus alphabetischen, numerischen und Sonderzeichen. Es gibt noch eine Reihe weiterer Zeichen, die alle angezeigt oder auch ausgedruckt werden können. Diese haben jedoch für Amiga Basic keine spezielle Bedeutung.

Die alphabetischen Zeichen von Amiga Basic enthalten alle Buchstaben des englischen Alphabets. Die numerischen Zeichen sind die Ziffern 0 bis 9. Die Sonderzeichen sind zusammen mit ihrer Bedeutung in der folgenden Tabelle zusammengestellt:

Zeichen	Bedeutung
	Leerzeichen
+	Plussymbol oder Verkettungssymbol für Zeichenketten
–	Minussymbol oder Bindestrich
*	Multiplikationssymbol oder Stern
/	Divisionssymbol oder Schrägstrich
\	Symbol für Ganzzahldivision oder linker Schrägstrich
↑	Potenzierungssymbol
=	Gleichheitszeichen oder Zuweisungssymbol
<	Kleiner–als–Symbol oder linke spitze Klammer
>	Größer–als–Symbol oder rechte spitze Klammer
(	linke Klammer
)	rechte Klammer
[	linke eckige Klammer
]	rechte eckige Klammer
%	Typsymbol für kurze Ganzzahl oder Prozentzeichen
&	Typsymbol für lange Ganzzahl oder kaufmännisches Und
#	Typsymbol für Zahlen doppelter Genauigkeit oder Nummernzeichen
$	Typsymbol für Zeichenketten oder Dollarzeichen
!	Typsymbol für Zahlen einfacher Genauigkeit oder Ausrufungszeichen
	Dezimalpunkt oder Punkt
	Komma

Zeichen	Bedeutung
:	Trennzeichen für Befehle oder Doppelpunkt
;	Zeilenvorschubunterdrückung oder Semikolon
'	Kommentareinleitung oder Apostroph
"	Zeichenketten–Anfang bzw. –Ende oder Anführungszeichen
?	Abkürzung für PRINT–Anweisung oder Fragezeichen
_	Unterstreichung
@	At–Zeichen ("Klammeraffe")
RETURN	Zeilenabschlußzeichen, Wagenrücklauf

Alle diese Zeichen können auch als normale druckbare Zeichen Bestandteil von Zeichenketten sein. Weitere druckbare Zeichen sind in der vollständigen ASCII-Zeichencodetabelle im Anhang A zusammengestellt.

Mit Hilfe der **rechten** Amiga–Taste können weitere Spezialzeichen eingegeben werden, die für Amiga Basic besondere Bedeutung haben:

Tastenkombination	Bedeutung
Amiga–. (Punkt)	Unterbricht ein Amiga Basic–Programm und kehrt auf Befehlsebene zurück.
Amiga–S	Hält ein Amiga Basic–Programm an.
Amiga–T	Führt die nächste Programmanweisung aus.
Amiga–C	Führt die Edierfunktion Copy (Kopiere) aus.

Tastenkombination	Bedeutung
Amiga–P	Führt die Edierfunktion Paste (Einfügen) aus.
Amiga–X	Führt die Edierfunktion Cut (Löschen) aus.
Amiga–R	Führt die Funktion Start aus dem Run–Menü aus.
Amiga–L	Führt die Funktion Show List (List–Fenster anzeigen) aus dem Windows–Menü aus.

8.2 Die Amiga Basic–Programmzeile

Amiga Basic–Programmzeilen haben das folgende Format:

[*nnnnn*] *Anweisung* [:*Anweisung*. . .]['*Kommentar*]

oder

[*Marke*:] *Anweisung* [:*Anweisung*. . .]['*Kommentar*]

nnnnn gibt die Zeilennummer an und muß eine ganze Zahl zwischen 0 und 65529 sein. Wie *Marke* (s. unten) stellt die Zeilennummer eine Sprungmarke für Verzweigungsoperationen dar. Sie hat **keinen** Einfluß auf die Reihenfolge der Programmzeilen.

Marke: ist eine beliebige Kombination aus alphanumerischen, numerischen und Sonderzeichen bis zu einer Länge von 40 Zeichen, die mit einem Buchstaben anfangen und mit dem Doppelpunkt (:) beendet werden muß. Sie stellt eine Sprungmarke für Verzweigungsanweisungen dar.

Anweisung ist ein beliebiger Amiga Basic–Befehl, eine –Anweisung, oder –Funktion. Mehrere von diesen in einer Zeile müssen durch Doppelpunkte (:) voneinander getrennt werden.

Kommentar kann ein bliebiger Text zur Kommentierung der Programmzeile sein. Er wird entweder eingeleitet durch ein Apostroph (') oder durch das Schlüsselwort REM mit vorangestelltem Doppelpunkt, also :REM (s.a. REM–Anweisung in Kapitel 9).

Amiga Basic–Programmzeilen können also mit einer Zeilennummer, mit einer alphanumerischen Sprungmarke, mit beiden oder ohne beide beginnen. Sie müssen jedoch mit einem Wagenrücklaufcode (RETURN) beendet werden und dürfen höchstens 255 Zeichen lang sein. Weitere Programmzeilen werden in ein vorhandenes Programm eingefügt, indem der Cursor auf das Ende der vorhergehenden Zeile gesetzt und die RETURN–Taste gedrückt wird. Es wird dann eine Leerzeile eingefügt, an deren Anfang der Cursor gestellt wird. Der Wagenrücklaufcode ist der einzige unsichtbare Bestandteil einer Amiga Basic–Programmzeile.

Zeilennummern und Sprungmarken dienen entweder zur Strukturierung eines Programmes, also dazu, es übersichtlicher zu machen, oder als Markierungen für die Programmsteuerung, also als Sprungziele für sämtliche Verzweigungsanweisungen (s. GOTO, GOSUB).

Wenn Sie z.B. einen besonderen Programmteil nur unter einer bestimmten Bedingung abarbeiten wollen, könnten Sie z.B. schreiben:

```
IF BETRAG=0 THEN GOSUB STORNO
```

Der Amiga Basic–Interpreter sucht dann nach einer Sprungmarke namens STORNO: und bearbeitet die damit beginnene Subroutine. Beachten Sie, daß der Doppelpunkt nur bei der Sprungmarke selbst stehen muß, nicht bei ihrer Verwendung in einer Verzweigungsanweisung.

Achtung: Der Amiga Basic–Interpreter arbeitet die Zeilen in einem Programm in der Reihenfolge ab, in der sie eingegeben wurden und **nicht** in der Reihenfolge ggf. vergebener Zeilennummern. Dies ist ein wesentlicher Unterschied zu herkömmlichen Basic–Dialekten!

8.3 Definition von Sprungmarken

Wie bereits erwähnt, dürfen alphanumerische Sprungmarken zwischen 1 und 40 Zeichen lang sein, müssen mit einem Buchstaben beginnen und mit einem Doppelpunkt (:) enden, können sonst beliebige Zeichen enthalten und müssen auf jeden Fall am Anfang einer Zeile stehen. Ihre Verwendung macht neben der Programmsteuerung die Lesbarkeit von Progammen leichter.

Folgende Beispiele für Zeilennummern und Sprungmarken sind gültig:

Zeilennummern	Sprungmarken
100	ALPHA:
65529	Beta:
1	Sprung.Ziel:

Einschränkungen

Der Doppelpunkt am Ende einer Sprungmarke ist deshalb erfoderlich, damit der Interpreter Sprungmarken und Variablennamen auseinanderhalten kann. Es darf deshalb auch keine Leerstelle zwischen Sprungmarke und Doppelpunkt stehen. Bei der Verwendung einer Sprungmarke in einer Verzweigungs– oder sonstigen Steueranweisung darf der Doppelpunkt jedoch nicht angegeben werden. Es dürfen außerdem keine reservierten Amiga Basic–Wörter (s. Anhang C) für Sprungmarkennamen verwendet werden.

Während die Zahl 0 als Zeilennummer in einem Programm erlaubt ist, wird sie von den Unterbrechungsreaktionsanweisungen zum Abschalten der Reaktionsfähigkeit verwendet:

```
ON ERROR GOTO 0
```

bedeutet deshalb nicht eine Verzweigung zur Zeile 0, sondern vielmehr, daß die Unterbrechungsreaktionsfähigkeit bei Programmfehlern abgeschaltet wird.

Achtung: Amiga Basic verwendet Zeilennummern ausschließlich als Sprungmarken. Sie werden nicht sortiert und Doubletten sind möglich.

Format

Eine Sprungmarke, eine Zeilennummer oder beides kann in jeder beliebigen Programmzeile vorkommen. Wird eine Zeilennummer angegeben, so muß sie in der äußerst linken Spalte beginnen. Eine Sprungmarke muß mit dem ersten von einer Leerstelle verschiedenen Zeichen, das auf die Zeilennummer (falls vorhanden) folgt, beginnen und muß mit einem Doppelpunkt enden. Zwischen Sprungmarke und Doppelpunkt darf kein Leerzeichen stehen.

Sowohl alphanumerische Sprungmarken als auch Zeilennummern können im selben Programm vorkommen.

8.4 Konstanten

Konstanten sind die aktuellen Werte, die Amiga Basic während der Programmausführung benutzt. Es gibt zwei Konstanten–Typen.

- Zeichenkettenkonstanten
- numerische Konstanten

Eine Zeichenkettenkonstante ist eine Folge von bis zu 32767 alphanumerischen Zeichen, eingeschlossen in Anführungszeichen ("). Z.B.:

```
"HAUS"
"Hund und Katze"
"DM 12.80"
```

Numerische Konstanten sind positive oder negative Zahlen. Amiga Basic unterscheidet 6 Typen numerischer Konstanten:

- kurze Ganzzahl — Alle ganzen Zahlen zwischen –32768 und +32767 ohne Dezimalpunkt.
- lange Ganzzahl — Alle ganzen Zahlen zwischen –2147483648 und +2147483647 ohne Dezimalpunkt.
- Festpunktzahl — Alle positiven oder negativen reellen Zahlen mit einem Dezimalpunkt.
- Gleitpunktzahl — Alle positiven oder negativen Zahlen in Exponentialdarstellung (wissenschaftliche Darstellung). Sie bestehen aus einer ganzen oder einer Festpunktzahl (Mantisse) mit oder ohne Vorzeichen, gefolgt vom Buchstaben D (doppelte Genauigkeit) oder E (einfache Genauigkeit) und einer ganzen Zahl im Bereich zwischen –308 und +308 mit oder ohne Vorzeichen, die den Exponenten zu Basis 10 darstellt.
- Hexadezimalzahl — Alle Zahlen in hexadezimaler Darstellung. Sie dürfen bis zu vier Zeichen (Ziffern 0 bis 9; Buchstaben A bis F) enthalten, denen die Kennung &H vorangestellt ist.
- Oktalzahl — Alle Zahlen in oktaler Darstellung. Sie dürfen bis zu 6 Ziffern im Bereich zwischen 0 und 7 enthalten, denen die Kennung &O oder nur & vorangestellt ist.

Fest– und Gleitpunktzahlen können mit einfacher oder doppelter Genauigkeit verarbeitet werden. Bei einfacher Genauigkeit werden diese Zahlen mit bis zu 7 Stellen plus dem Exponenten gespeichert und mit bis zu 7 Stellen angezeigt oder ausgedruckt. Bei doppelter Genauigkeit werden diese Zahlen mit bis zu 16 Stellen plus dem Exponenten gespeichert und mit bis zu 16 Stellen angezeigt oder ausgedruckt. Im Anhang D wird die interne Darstellung von Zahlen bei Amiga Basic detailliert beschrieben.

Eine numerische Konstante einfacher Genauigkeit hat eine der folgenden Eigenschaften:

- 7 oder weniger Stellen
- Exponentialdarstellung mit Exponentkennung E
- Ein Ausrufungszeichen (!) am Ende

Eine numerische Konstante doppelter Genauigkeit hat eine der folgenden Eigenschaften:

- 8 oder mehr Stellen
- Exponentialdarstellung mit Exponentkennung D
- Ein Nummernzeichen (#) am Ende

Hier einige Beispiele für numerische Konstanten:

`1234`	kurze Ganzzahl
`15.12` `-0.56`	Festpunktzahl
`2345678` `-45789369`	lange Ganzzahlen
`15E-3` `-1.24683D-4`	Gleitpunktzahlen
`&H2DEF` `-&HFFFF`	Hexadezimalzahlen
`&0177` `-&336`	Oktalzahlen

Numerische Konstanten in Amiga Basic dürfen keine Kommata enthalten.

8.5 Variablen

Variablen sind Namen, die Werte repräsentieren, welche in einem Amiga Basic-Programm verwendet werden. Der Interpreter unterscheidet dabei zwischen

- Zeichenketten–Variablen
- numerischen Variablen

Zeichenkettenvariablen dürfen aus bis zu 32767 Zeichenkettenwerten (einzelne Zeichen) bestehen. Die Anzahl dieser Werte bestimmt die Länge des Variablenwertes.

Numerische Variablen haben immer einen Wert, der aus einer Zahl besteht.

Der Wert einer Variablen kann dieser explizit vom Programmierer oder, als Ergebnis von Programmoperationen, vom Programm zugewiesen werden. Werden Variablen benutzt, ehe ihnen ein Wert zugewiesen wurde, ist ihr Wert 0 im Fall von numerischen Variablen und eine leere Zeichenkette der Länge 0 (Leerstring) im Fall von Zeichenkettenvariablen.

Variablennamen und –Typen

Amiga Basic akzeptiert beliebig lange Variablennamen, interpretiert jedoch nur die ersten 40 Zeichen sowie ggf. das letzte, das den Variablentyp definiert (s. unten) und eines der Sonderzeichen $ % & ! # sein muß, wenn ein anderer Typ als numerisch mit einfacher Genauigkeit definiert werden soll.

Ein Variablenname muß als erstes Zeichen einen Buchstaben haben und darf aus Buchstaben, Ziffern und dem Dezimalpunkt in jeder beliebigen Kombination bestehen. Kleinbuchstaben werden intern als Großbuchstaben interpretiert. Er darf nicht ein reserviertes Amiga Basic–Wort (s. Anhang C) sein; wohl aber darf ein reserviertes Wort Bestandteil des Namens sein: Z.B.:

```
SIN=101          verboten (SIN–Funktion)
SINPUT=101       erlaubt (SIN nur Teil des Namens)
```

Eine Ausnahme bilden Namen, die mit FN beginnen. Der Teil nach FN würde nämlich von Amiga Basic als benutzerdefinierte Funktion interpretiert (s. DEF FN–Anweisung in Kapitel 9).

Der Name einer Variablen dient nicht nur ihrer Benennung, sondern auch der Definition ihres Typs und, falls numerisch, ihrer Genauigkeit.

Zeichenkettenvariablen werden durch ein $ als letztem Zeichen im Namen definiert. Z.B.:

```
C$="COMMODORE AMIGA"
```

Numerische Ganzzahlvariablen werden durch ein % oder & als letztem Zeichen im Namen definiert, je nachdem, ob es eine kurze oder lange Ganzzahlvariable ist. Z.B.:

```
TAG%=30:GEWINN&=456789258
```

Numerische Variablen einfacher Genauigkeit haben kein Sonderzeichen oder ein ! am Namensende. Z.B.:

```
ZINS=15.84
ZINS!=15.84
```

Numerische Variablen doppelter Genauigkeit werden durch ein # als letztem Zeichen im Namen definiert. Z.B.:

```
ERG#=45.789678524
```

Je größer die geforderte Genauigkeit ist, umso mehr Speicher und Programmlaufzeit werden benötigt. Der Platzbedarf ist bei:

kurzen Ganzzahlvariablen	2 Bytes
langen Ganzzahlvariablen	4 Bytes
Variablen einfacher Genauigkeit	4 Bytes (7 Stellen)
Variablen doppelter Genauigkeit	8 Bytes (16 Stellen)
Zeichenkettenvariablen	5 Bytes plus Länge der Zeichenkette

Eine globale Variablen–Typdefinition kann durch die Anweisungen DEFINT, DEFLNG, DEFSNG, DEFDBL und DEFSTR (s. Kapitel 9) in einem Programm vorgenommen werden.

Feldvariablen

Ein Feld ist eine Gruppe oder Tabelle von numerischen oder Zeichenkettenwerten, die durch einen Variablennamen repräsentiert wird. Jeder dieser Werte wird als Feldelement bezeichnet und mit einem Index zum Variablennamen beschrieben. Dieser Index ist einen ganze Zahl oder ein ganzzahliger Ausdruck. Einem Feldvariablennamen sind genausoviele Indizes zugeordnet, wie das Feld Dimensionen hat. Die Dimensionierung eines Feldes, mit der gleichzeitig auch die Benennung verbunden ist, erfolgt mit der Anweisung DIM (s. dort in Kapitel 9). Z.B.:

```
DIM X(25)
```
Ein eindimensionales Feld X mit 26 Elementen (0 bis 25) einfacher Genauigkeit wird dimensioniert.

```
DIM X$(2,6)
```
Ein zweidimensionales Zeichenkettenfeld mit 21 Elementen (3*7) wird dimensioniert.

Mit der Dimensionierung werden alle Feldelemente des dimensionierten Feldes auf Null gesetzt.

Der Index bestimmt die Position eines Elementes im Feld.

Die maximale Anzahl von Elementen in jeder Dimension ist 32767 und es sind maximal 255 Dimensionen möglich.

Bei zweidimensionalen Feldern gibt der erste Index grundsätzlich die Zeile und der zweite Index die Spalte der durch ein solches Feld gebildeten Tabelle an.

Eindimensionale Felder mit bis zu 11 Elementen brauchen nicht dimensioniert zu werden. Wird in einem Programm z.B. die Anweisung

```
A(6)=12.3
```

verwendet, ohne daß vorher A dimensioniert wurde, so führt der Interpreter intern die Anweisung

```
DIM A(10)
```

aus und dimensioniert damit das Feld selbst.

Für die Namensvergabe, Genauigkeit und den Typ gelten dieselben Regeln wie für einfache Variablen (s. oben).

Wandlung von einer Genauigkeit in eine andere

Bei der Zuordnung von Werten zu Variablen einer anderen Genauigkeit sind einige Regeln zu beachten, nach denen der Interpreter dabei verfährt. Im einzelnen sind dies:

1. Wird ein numerischer Wert einer Variablen mit anderer Genauigkeit zugeordnet, so wird der Wert in der Genauigkeit der Variablen gespeichert und dabei ggf. gerundet. Z.B.:

```
X%=12.5:PRINT X%
 13
Ok
```

2. Wird ein Wert höherer Genauigkeit einer Variablen niedrigerer Genauigkeit zugeordnet, so wird er gerundet. Z.B.:

```
A=12.75455835#:PRINT A
 12.75456
Ok
```

   Dies ist insbesondere dann zu beachten, wenn gebrochene Werte angegeben werden, wo ganzzahlige gefordert sind. Z.B.:

   `TAB(8.5)` ergibt `TAB(9)`
   `B(3.6)` ergibt `B(4)`

3. Wird ein Wert niedrigerer Genauigkeit einer Variablen höherer Genauigkeit zugeordnet, so kann er danach nicht genauer sein als vorher. Der Fehler, der dabei entsteht, liegt etwa bei 6.3E–8. Z.B.:

```
A=1.65:B#=A:PRINT A;B#
 1.65  1.649999976158142
Ok
```

4. Die Berechnung arithmetischer Ausdrücke erfolgt immer in der Genauigkeit des genauesten Operanden. In dieser Genauigkeit wird auch das Ergebnis geliefert. Z.B.:

```
PRINT 5#/9
. 5555555555555556
Ok
A=5#/9:PRINT A
. 5555556
Ok
```

Im letzten Beispiel wird das doppeltgenaue Ergebnis der einfachgenauen Variablen A zugeordnet und damit gerundet (s. Regel 2).

5. Logische Operatoren (s. Kapitel 8.6) wandeln ihre Operanden grundsätzlich in ganzzahlige Werte um und liefern auch ganzzahlige Ergebnisse. Die Operanden müssen deshalb auch im Bereich zwischen –32768 und 32767 liegen.

8.6 Numerische Ausdrücke und Operatoren

Ein numerischer Ausdruck kann eine numerische Konstante oder Variable oder eine Kombination aus numerischen Konstanten, Variablen und Operatoren zur Berechnung eines einzelnen Wertes sein. Die Operatoren führen dabei mit den Werten mathematische oder logische Operationen durch. Amiga Basic kennt vier verschiedene Kategorien von Operatoren:

- arithmetische Operatoren
- Vergleichsoperatoren
- logische Operatoren
- Funktionsoperatoren

Mathematische Hierarchie

Bei der Bearbeitung numerischer Ausdrücke und Operatoren hält Amiga Basic folgende mathematische Hierarchie ein (1 höchste, 12 niedrigste Priorität):

1. Funktionsaufrufe
2. Potenzierung (↑)
3. Negation (Wandlung von Plus nach Minus u. umgekehrt)
4. Multiplikation und Division (*, /)
5. Ganzzahldivision (\)
6. Modulo–Arithmetik (MOD)
7. Addition und Subtraktion (+, –)
8. logisches Komplement (NOT)
9. Konjunktion (AND)

10. Disjunktion (OR)
11. exklusives ODER (XOR)
12. Äquivalenz (EQV)
13. Implikation (IMP)

Aufeinanderfolgende Operatoren derselben Hierarchiestufe werden von links nach rechts abgearbeitet. Die Hierarchie kann durch Verwendung von Klammern aufgehoben werden. Geklammerte Ausdrücke werden grundsätzlich zuerst bearbeitet. Bei geschachtelten Klammerausdrücken wird grundsätzlich der innerste Klammerausdruck zuerst bearbeitet.

Arithmetische Operatoren

Es gibt acht arithmetische Operatoren. Es sind dies in der Reihenfolge ihrer Berücksichtigung (mathematische Hierarchie) in einem numerischen Ausdruck:

Operator	Operation	Beispiel
↑	Potenzierung	A ↑ 5
–	Negation	–A
*,/	Multiplikation, Gleitpunktdiv.	A*5, A/5
\	Ganzahldivision	A\5
MOD	Modulo–Arithmetik	A MOD 5
+,–	Addition, Subtraktion	A+5, A–5

Die Ganzzahldivision und die Modulo–Arithmetik sind Besonderheiten des mathematischen Teils des Amiga Basic–Interpreters.

Bei der **Ganzahldivision**, bei der Divisor und Dividend im Bereich zwischen –32768 und +32767 im Fall von kurzen Ganzzahlen und zwischen –2147483648 und +2147483647 im Fall von langen Ganzzahlen liegen müssen, werden beide vor der Division zu ganzen Zahlen gerundet. Bei Quotienten werden eventuelle Dezimalstellen abgeschnitten. Z.B.:

```
PRINT 7\3,12.9\5.89
 2 2
Ok
```

Bei der **Modulo–Arithmetik**, die eine ganzzahlige Division durchführt, werden zunächst beide Operanden, die im Bereich zwischen –32768 und +32767 liegen müssen, zu ganzen Zahlen gerundet. Als Ergebnis wird hier jedoch der Rest der Division geliefert. Z.B.:

```
PRINT 7 MOD 2; 12.9 MOD 5.89
 1  1
Ok
```

Vergleichsoperatoren

Vergleichsoperatoren dienen dem Vergleich zweier numerischer oder Zeichenkettenwerte. Das Ergebnis ist entweder logisch "wahr" mit dem Wert –1 oder logisch "falsch" mit dem Wert 0 (Null) und wird meistens in Verbindung mit der IF–Anweisung (s. dort in Kapitel 9) zur Steuerung des Programmablaufes verwendet.

Hier sind die sechs verschiedenen Vergleichsoperatoren, die Amiga Basic kennt, tabellarisch zusammengestellt:

Operator	Vergleich auf	Beispiel
=	Gleichheit	A=B
<>, ><	Ungleichheit	A<>B, A><B
<	kleiner als	A<B
>	größer als	A>B
<=, =<	kleiner als oder gleich	A<=B, A=<B
>=, =>	größer als oder gleich	A>=B, A=>B

Wenn in einem Ausdruck sowohl arithmetische als auch Vergleichsoperatoren vorkommen, so werden zuerst die arithmetischen Operatoren abgearbeitet. Z.B.:

```
A+B<(X-Y)/Z
```

Dieser Ausdruck ist "wahr" (liefert den Wert –1), wenn der Wert von A+B kleiner ist als der Wert X–Y dividiert durch Z.

Oder z.B.:

```
100 IF SIN(X)<0 GOTO 1000
```

Wenn der SIN(X) negativ wird, liefert der Vergleich ein "wahres" Ergebnis (–1) und das Programm verzweigt nach Zeile 1000.

Bei Zeichenkettenvergleichen wird von beiden Ketten zeichenweise der ASCII–Code verglichen. Es werden alle Zeichen, also auch führende und nachfolgende Leerstellen verglichen. Die Wirkung ist am besten an folgenden Beispielen zu erkennen, die alle logisch "wahr" sind, also den Wert –1 liefern:

```
"AA"<"AB"
"Aa">"AA"
"HAUS"="HAUS"
"HAUS">"HANS"
"A!"<"A$"
"U ">"U"
X$="12":X$<"13"
```

Logische Operatoren

Logische Operatoren dienen zum Testen von Mehrfachvergleichen, zur Bit-Manipulation oder zum Durchführen Boole'scher Operationen mit numerischen Werten.

Ein logischer Operator verknüpft zwei Operanden als Kombination aus "wahr"– und "falsch"–Werten bitweise und liefert als Ergebnis einen Wert, der entweder als "wahr" (von Null verschieden) oder als "falsch" (Null) interpretiert wird.

Der Interpreter führt Operationen mit logischen Operatoren in einem gemischten Ausdruck **nach** den arithmetischen und den Vergleichsoperationen durch.

Der Interpreter kennt sechs verschiedene logische Operatoren, die im folgenden in der Reihenfolge, wie sie bearbeitet werden, mit ihren Wahrheitstabellen dargestellt werden:

Operator	**Operand 1**	**Operand 2**	**Ergebnis**
NOT (log. Komplement)	wahr	–	falsch
	falsch	–	wahr
AND (Konjunktion)	wahr	wahr	wahr
	wahr	falsch	falsch
	falsch	wahr	falsch
	falsch	falsch	falsch
OR (Disjunktion)	wahr	wahr	wahr
	wahr	falsch	wahr
	falsch	wahr	wahr
	falsch	falsch	falsch
XOR (exkl. ODER)	wahr	wahr	falsch
	wahr	falsch	wahr
	falsch	wahr	wahr
	falsch	falsch	falsch
EQV (Äquivalenz)	wahr	wahr	wahr
	wahr	falsch	falsch
	falsch	wahr	falsch
	falsch	falsch	wahr
IMP (Implikation)	wahr	wahr	wahr
	wahr	falsch	falsch
	falsch	wahr	wahr
	falsch	falsch	wahr

Genau wie die Vergleichsoperatoren über ihr Ergebnis zur Steuerung des Programmablaufes beitragen können, kann dies auch durch die Verknüpfung von zwei oder mehreren Vergleichen durch logische Operatoren geschehen, die ja wiederum "wahr"– und "falsch"–Werte liefern (s. a. IF–Anweisung in Kapitel 9).

Die folgenden Beispiel–Programmzeilen sollen dies verdeutlichen:

```
IF D<200 AND F<4 THEN 80
IF I>10 OR K<0 THEN 50
IF NOT P THEN 100
```

Die logischen Operatoren arbeiten intern folgendermaßen:

Zunächst werden die beiden Operanden in ganze, vorzeichenbehaftete Zweierkomplement–16–Bit–Zahlen im Bereich zwischen –32768 und +32767 umgewandelt. Sind die Operanden größer oder kleiner, so wird die Fehlermeldung

`Overflow error` (Überlauf)

angezeigt. Sind die beiden Operanden 0 und/oder –1, so liefert eine logische Operation ebenfalls 0 oder –1. Die Operation wird auf jeden Fall bitweise durchgeführt, d. h. jedes Ergebnis–Bit wird durch die beiden entsprechenden Bits in den Operanden bestimmt. Dabei bedeutet ein 1–Bit "wahr" und ein 0–Bit "falsch". Dadurch ist es möglich, mit Hilfe der logischen Operatoren das Bitmuster von Speicherzellen zu testen. Z.B. kann ein Byte an einer Ein–/Ausgabeschnittstelle maskiert werden, um den Zustand eines bestimmten Bits zu testen. Auch kann mit dem OR–Operator in einem bestimmten Byte ein ganz bestimmtes Bitmuster erzeugt werden. Die folgenden Beispiele sollen die Arbeitsweise der logischen Operatoren erläutern:

```
63 AND 16 = 16          0000000000111111    63
                AND     0000000000010000    16
                        0000000000010000 =  16

15 AND 14 = 14          0000000000001111    15
                AND     0000000000001110    14
                        0000000000001110 =  14

–1 AND 8 = 8            1111111111111111    –1
                AND     0000000000001000    8
                        0000000000001000 =  8

4 OR 2 = 6              0000000000000100    4
                OR      0000000000000010    2
                        0000000000000110 =  6
```

10 OR 10 = 10		0000000000001010	10
	OR	0000000000001010	10
		0000000000001010 =	10
–1 OR –2 = –1		1111111111111111	–1
	OR	1111111111111110	–2
		1111111111111111 =	–1
NOT 1 = –2		0000000000000001	1
	NOT	1111111111111110 =	–2

Der Operator NOT bildet das Einerkomplement des Operanden.

Funktionsoperatoren

Funktionsoperatoren (Amiga Basic–Funktionen) werden dazu verwendet, an einem Operanden eine festgelegte Operation auszuführen. Solche in Amiga Basic enthaltenen Funktionen sind z.B. SQR (Quadratwurzel ziehen) oder SIN (trigonometrischer Sinus). Es ist auch möglich, mit Hilfe der DEF FN–Anweisung eigene Funktionen zu definieren. Einzelheiten zu den Funktionen sind in Kapitel 9 ausführlich beschrieben.

8.7 Operatoren bei Zeichenketten

Ein Zeichenkettenausdruck kann eine Zeichenkettenkonstante oder –Variable oder eine Kombination aus Zeichenkettenkonstanten, –variablen und –operatoren zur Erzeugung einer neuen Zeichenkette sein. Die Operatoren kombinieren dabei Zeichenketten zu neuen Zeichenketten. Amiga Basic kennt drei Typen von Zeichenkettenoperatoren:

- Verkettungsoperator
- Vergleichsoperatoren
- Funktionsoperator oder Zeichenkettenfunktion

Da der Zeichenkettenvergleich prinzipiell ein numerischer Vergleich ist, ist er im letzten Kapitel bei den Vergleichsoperatoren beschrieben.

Verkettungsoperator

Zeichenketten können mit dem +–Zeichen aneinandergefügt werden, um neue Zeichenketten zu bilden. Z.B.:

```
A$="REGEN":B$="SCHIRM"
PRINT A$+B$+"E"
REGENSCHIRME
Ok
```

Zeichenkettenfunktionen

Wie bei den numerischen Funktionen (s. Kapitel 8.6) führt die Zeichenkettenfunktion an einem oder mehreren Operanden eine festgelegte Operation aus, die als Ergebnis eine Zeichenkette liefert. Solche in Amiga Basic enthaltenen Funktionen sind z.B. CHR$ (Bildung einer Ein–Zeichenkette aus einem ASCII–Code) oder STR$ (Bildung eines Zeichenkettenäquivalents zu einem numerischen Ausdruck). Es ist auch möglich, eigene Zeichenkettenfunktionen mit Hilfe der DEF FN–Anweisung zu definieren. Einzelheiten zu den Zeichenkettenfunktionen sind in Kapitel 9 ausführlich beschrieben.

9. Amiga Basic–Befehle, –Anweisungen, –Funktionen und –Variablen

9.1 Struktur von Amiga Basic

Der Sprachumfang der Programmiersprache Amiga Basic gliedert sich in:

- Befehle
- Anweisungen
- Funktionen
- Variablen

Befehle werden dazu benutzt, an Programmen etwas zu bearbeiten, sie zu listen, zu ändern, zu löschen, zu speichern oder zu laden. Deshalb werden Befehle auch meistens in der Befehlsebene (Direktmodus) eingegeben. Sie können aber auch Bestandteil eines Programms sein.

Anweisungen steuern den Programmablauf. Sie sind der eigentliche Inhalt des Programms, können aber auch im Direktmodus eingegeben werden. Die Anweisungen gliedern sich in 4 Gruppen:

- Ein–/Ausgabeanweisungen für den Datenverkehr zwischen dem Rechner einerseits sowie Tastatur, Bildschirm, Diskettenstation, Festplatte, Drucker oder Datenfernübertragungsschnittstelle andererseits.
- Grafische, akustische und Animations–Anweisungen zur Nutzung der herausragenden audiovisuellen Eigenschaften des Amiga.
- Deklarationsanweisungen für die Definition von Daten, benutzereigenen Funktionen, Speichersegmenten oder der Programmumgebung.
- Steueranweisungen zur Festlegung des Programmablaufes wie Schleifen, Sprünge, Unterprogramm– und Unterbrechungsaufrufe.

Funktionen liefern als Ergebnis bestimmte Werte zur Weiterverarbeitung an das Programm und gliedern sich in:

- Numerische Funktionen, die als Ergebnis Werte in gewünschter Genauigkeit übergeben;
- Zeichenkettenfunktionen, die als Ergebnis eine Zeichenkette übergeben.

Variablen des Amiga Basic dürfen nicht mit den vom Programmmierer frei definierbaren Variablen eines Programms verwechselt werden. Die hier gemeinten System–Variablen haben feste, von Basic vorgegebene Namen und Funktionen. Ihnen wird beim Aufruf vom Interpreter ein Wert zugewiesen, der bestimmte Funktionszustände des Interpreters oder Rechnersystems zu einem bestimmten Zeitpunkt beschreibt. Es gibt numerische und Zeichenketten–Systemvariablen.

9.2 Struktur und Syntax der Beschreibungen

Im Kapitel 9.3 werden alle Befehle, Anweisungen, Funktionen und Variablen, die Amiga Basic kennt, detailliert und an Hand von Beispielen beschrieben. Die einzelnen Beschreibungen sind alphabetisch nach den Namen geordnet und nach dem folgenden Schema gegliedert:

Format: Hier ist die exakte Schreibweise für einen Befehl, eine Anweisung, Funktion oder Variable angegeben. Dabei gelten die folgenden Syntax–Regeln:

1. Alle Wörter in Großbuchstaben sind Amiga Basic–Schlüsselwörter, deren Schreibweise wie angegeben verbindlich ist. Bei der Eingabe über Tastatur können dabei Klein– oder Großbuchstaben verwendet werden, da der Interpreter diese Schlüsselwörter generell in Großbuchstaben umwandelt.

2. Alle Angaben in kursiver Groß– und Kleinschrift stellen Parameter dar, die vom Anwender eingesetzt werden müssen.

3. Angaben zwischen eckigen Klammern ([]) sind wahlfrei.

4. Eine Folge von Punkten (. . .) hinter einer Angabe bedeutet, daß diese Angabe, falls erforderlich, mehrmals spezifiziert werden kann.

5. Alle Sonderzeichen außer den eckigen Klammern, also , ; – = ( ) ' , müssen da, wo sie stehen, auch angegeben werden.

6. Bei den Parameterangaben haben die dort verwendeten Abkürzungen folgende Bedeutung:

x,y,z	beliebige numerische Ausdrücke
i,j,k,m,n	beliebige ganzzahlige Ausdrücke
x$,y$	beliebige Zeichenkettenausdrücke
v,v$	numerische oder Zeichenkettenvariablen

Bedeutung: Hier wird die Wirkung des jeweiligen Befehls oder der Anweisung, Funktion oder Variablen detailliert beschrieben, und die Parameter und ihre Definitionsbereiche werden erläutert.

Beispiel: Ein oder mehrere Beispiele bei jeder Beschreibung zeigen, wie die einzelnen Befehle, Anweisungen, Funktionen und Variablen angewendet werden, und zeigen auch Besonderheiten auf.

Anmerkungen: Besondere Eigenschaften der einzelnen Befehle, Anweisungen, Funktionen und Variablen werden hier erläutert.

9.3 Beschreibung der einzelnen Befehle, Anweisungen, Funktionen und Variablen

Zur Erleichterung des Auffindens eines bestimmten Amiga Basic–Befehls oder einer –Anweisung, –Funktion oder –Variablen sind die nachfolgenden Beschreibungen nicht entsprechend der in Kapitel 9.1 beschriebenen Sprachstruktur von Amiga Basic gegliedert, sondern es sind alle Sprachelemente zusammengefaßt und alphabetisch geordnet.

Für den schnellen Überblick sind den einzelnen Beschreibungen zunächst nach Sprachelementen geordnet die einzelnen Befehle, Anweisungen, Funktionen und Variablen mit einem kurzen Stichwort über ihre Wirkung in alphabetischer Reihenfolge in Listen vorangestellt. Dabei sind jedoch in den meisten Fällen alle Parameter weggelassen.

Befehle

CHDIR	Ein anderes als das aktuelle Diskverzeichnis wird aktualisiert.
CLEAR	Alle Programmvariablen werden gelöscht und es wird der Arbeitsspeicherbereich definiert.
CONT	Ein unterbrochenes Programm wird fortgesetzt.
DELETE	Bestimmte Programmzeilen werden gelöscht.
FILES	Der Inhalt des aktuellen Diskverzeichnisses wird auf dem Bildschirm dargestellt.
KILL	Eine bestimmte Disk–Datei wird gelöscht.
LIST	Bestimmte Programmzeilen werden auf dem Bildschirm oder einer anderen Ausgabeeinheit gelistet.
LLIST	Bestimmte Programmzeilen werden auf dem Drucker gelistet.
LOAD	Eine Programmdatei wird von einer Eingabeeinheit geladen und ggf. gestartet.
MERGE	Eine Programmdatei wird von einer Eingabeeinheit zum hauptspeicherresidenten Programm dazugeladen.
NAME	Eine Diskdatei wird umbenannt.
NEW	Das hauptspeicherresidente Programm und alle seine Variablen werden gelöscht.
RUN	Das hauptspeicherresidente Programm wird von einer gewünschten Zeile an gestartet.
SAVE	Das hauptspeicherresidente Programm wird in wählbarem Format auf Disk gespeichert.

Befehle (Fortsetzung)

SYSTEM Rückkehr aus Amiga Basic zum Arbeitstisch (Workbench).

TRON, TROFF Die Programmablaufverfolgung wird ein– bzw. ausgeschaltet.

Ein–/Ausgabeanweisungen

BEEP	Kurzes audiovisuelles Signal.
CLOSE	Eine Datei wird geschlossen.
CLS	Der Bildschirm wird gelöscht.
DATA	Es werden Konstanten im Speicher abgelegt, die mit READ gelesen werden sollen.
GET #	Es wird ein Satz aus einer Datei mit wahlfreiem Zugriff gelesen.
INPUT	Es werden Daten von der Tastatur gelesen.
INPUT #	Es werden Daten aus einer bestimmten, sequentiellen Datei gelesen.
LIBRARY	Es wird eine Bibliothek mit Maschinensprache–Unterprogrammen geöffnet.
LINE INPUT	Es werden alle über Tastatur eingegebenen Zeichen bis zur Return–Taste als eine Zeile gelesen. Es gibt keine Trennzeichen (z.B. Komma).
LINE INPUT #	Es werden alle Zeichen bis zum Wagenrücklaufcode als eine Zeile aus einer sequentiellen Datei gelesen.
LOCATE	Der Cursor wird an eine wählbare Bildschirmposition gestellt.
LPRINT	Es werden Daten auf dem Drucker ausgegeben.
LPRINT USING	Es werden Daten in einem wählbaren Format auf dem Drucker ausgegeben.
LSET	Es werden Daten linksbündig in Datenfeldern, die in einem Datenpuffer für wahlfreien Zugriff definiert sind, abgelegt

Ein–/Ausgabeanweisungen (Fortsetzung)

OPEN Es wird eine Datei oder eine Ein–/Ausgabeeinheit zur Datenein– oder –ausgabe eröffnet.

POKE Ein Byte wird an einer wählbaren Adresse in den Hauptspeicher geschrieben.

POKEL Eine lange Ganzzahl wird an einer wählbaren Adresse in den Hauptspeicher geschrieben.

POKEW Eine kurze Ganzzahl wird an einer wählbaren Adresse in den Hauptspeicher geschrieben.

PRINT Es werden Daten auf dem Bildschirm angezeigt.

PRINT USING Es werden Daten in einem wählbaren Format auf dem Bildschirm angezeigt.

PRINT # Es werden Daten in eine sequentielle Datei ausgegeben.

PRINT #, USING Es werden Daten in einem wählbaren Format in eine sequentielle Datei ausgegeben.

PUT # Es werden Daten aus einem Puffer für wahlfreien Zugriff in eine Datei geschrieben.

READ Es werden Daten aus DATA–Anweisungszeilen gelesen und Variablen zugewiesen

RSET Es werden Daten rechtsbündig in Datenfeldern, die in einem Datenpuffer für wahlfreien Zugriff definiert sind, abgelegt.

SCROLL Ein definierter, rechteckiger Bereich im aktuellen Ausgabefenster wird in eine wählbare Richtung gerollt.

WRITE Es werden Daten auf dem Bildschirm angezeigt.

WRITE # Es werden Daten in eine sequentielle Datei ausgegeben.

Graphische, akustische und Animations–Anweisungen

AREA	Definiert einen Punkt eines Polygons.
AREAFILL	Malt ein Polygon mit einem definierten Muster aus.
CIRCLE	Malt eine Ellipse mit wählbarem Radius und Mittelpunkt.
COLOR	Setzt Vorder– und Hintergrundfarbe.
GET	Liest grafische Darstellungen vom Bildschirm als binäre Information in ein Ganzzahlfeld.
LINE	Malt eine Linie oder ein Viereck.
OBJECT.AX/AY	Definiert die Beschleunigung eines grafischen Objektes.
OBJECT.CLIP	Definiert den Darstellungsbereich für grafische Objekte.
OBJECT.CLOSE	Gibt den für nicht mehr benötigte grafische Objekte reservierten Speicherplatz wieder frei.
OBJECT.HIT	Legt für ein grafisches Objekt fest, ob dessen Kollisionen Unterbrechungsereignisse sind oder nicht.
OBJECT.ON/OFF	Macht ein oder mehrere grafische Objekte sichtbar oder unsichtbar.
OBJECT.PLANES	Definiert zwei 8–Bit–Masken, mit denen die Farbdarstellung für ein bestimmtes Objekt beeinflußt werden kann.
OBJECT.PRIORITY	Setzt die Priorität, mit der ein grafisches Objekt gegenüber anderen Objekten vom System gezeichnet wird.
OBJECT.SHAPE	Definiert Form, Farben, Ort und weitere Attribute für ein grafisches Objekt.
OBJECT.START/STOP	Setzt angegebene grafische Objekte in Bewegung oder hält sie an.

Graphische, akustische und Animations–Anweisungen (Fortsetzung)

OBJECT.VX/VY	Definiert für ein angegebenes grafisches Objekt die Geschwindigkeit in X– oder Y–Richtung.
OBJECT.X/Y	Plaziert ein angegebenes grafisches Objekt bei einer definierten Position im Ausgabefenster.
PAINT	Malt eine geschlossene Fläche in einer wählbaren Farbe aus.
PALETTE	Definiert einen Farbton für eine Farbkennung.
PATTERN	Definiert ein Muster, mit dem Linien und/oder Flächen gemalt werden.
PRESET/PSET	Zeichnet an einer definierten Position im Ausgabefenster einen Bildpunkt.
PUT	Überträgt aus einem Ganzzahlfeld binäre Informationen als grafische Darstellung in das Ausgabefenster.
SAY	Gibt eine Phonem–Code–Zeichenkette als verständliche Sprache beliebiger Nationalität über einen wählbaren Tonkanal aus.
SCREEN	Definiert die Attribute für einen neuen oder schließt einen existierenden Bildschirm.
SOUND	Gibt einen Ton wählbarer Frequenz, Dauer und Lautstärke über einen wählbaren Ton–Kanal aus oder hält die Tonausgabe an.
WAVE	Definiert die Form von Tonwellen für einen angegeben Tonkanal.
WINDOW	Definiert, aktualisiert oder schließt ein Ausgabefenster.

Deklarationsanweisungen

DECLARE FUNCTION	Deklariert eine Maschinensprache-Routine aus einer Bibliothek als eine aufrufbare Funktion.
DIM	Die maximale Anzahl von Elementen für Feldvariablen wird festgelegt und Speicherplatz dafür reserviert.
ERASE	Feldvariablen und dafür reservierter Speicherplatz werden gelöscht.
ERROR	Bestimmte Fehler werden simuliert oder anwenderspezifische Fehlercodes definiert.
FIELD	Es wird Platz für Variableninhalte in einem Puffer für Dateien mit wahlfreiem Zugriff reserviert.
LET	Der Wert eines beliebigen Ausdrucks wird einer wählbaren Variablen zugewiesen.
MENU	Deklariert ein Anwender–Menü oder setzt die Menü–Leiste zurück.
MID$	Ein Teil einer Zeichenkette wird durch eine andere Zeichenkette ersetzt (s.a. MID$–Funktion).
OPTION BASE	Der kleinste Indexwert für alle definierten Feldvariablen wird festgelegt.
RANDOMIZE	Der Zufallszahlengenerator wird mit einem neuen Anfangswert gestartet
REM	Einzelne Kommentare oder ganze Kommentarzeilen werden ins Programm eingefügt.
WIDTH	Die Zeilenlänge und Druckzonenbreite werden für eine Ausgabeeinheit (Bildschirm, Drucker, Datenfernübertragungskanal) festgelegt.

Steueranweisungen

BREAK ON/OFF/STOP — Die Reaktionsfähigkeit bei Programmunterbrechung durch den Anwender wird aktiviert oder inaktiviert.

CALL — Ein Amiga Basic–, ein Maschinensprache–Unterprogramm oder eine Bibliotheksroutine werden aufgerufen, und es werden wahlweise Parameter übergeben.

CHAIN — Ein anderes Amiga Basic–Programm wird aufgerufen, und es werden Variablen übergeben oder es werden Programmüberlagerungen ausgeführt.

COLLISION ON,OFF,STOP — Die Unterbrechungsreaktionsfähigkeit für Objekt–Kollisionen untereinander oder mit Fensterbegrenzungen wird aktiviert oder inaktiviert.

COMMON — Wählbare Variablen werden an ein durch CHAIN (s. oben) aufgerufenes Programm übergeben.

END — Ein laufendes Amiga Basic-Programm wird nach der Schließung aller offenen Dateien beendet.

FOR. . .NEXT — Alle Programmzeilen zwischen der FOR– und der NEXT-Anweisung werden in einer wählbaren Anzahl von Durchläufen wiederholt ausgeführt.

GOSUB — Das Programm verzweigt in eine Subroutine mit wählbarer Zeilennummer oder Sprungmarke.

GOTO — Das Programm verzweigt zu einer wählbaren Programmzeile.

IF. .THEN. .ELSE — Das Programm verzweigt abhängig vom logischen Wahrheitsgehalt eines numerischen Ausdrucks in verschiedene Programmteile.

MENU ON/OFF/STOP — Die Unterbrechungsreaktionsfähigkeit bei Menübenutzung durch den Anwender wird aktiviert.

Steueranweisungen (Fortsetzung)

MOUSE ON/OFF/STOP Die Unterbrechreaktionsfähigkeit bei Bestätigung der Maus-Auswahltaste wird aktiviert oder inaktiviert.

ON BREAK GOSUB Eine Programmverzweigung zu einer wählbaren Programmzeile bei Programmunterbrechung durch den Anwender wird ausgeführt.

ON COLLISION GOSUB Eine Programmverzweigung zu einer wählbaren Zeile bei Programmunterbrechung durch Objekt-Kollision wird ausgeführt.

ON ERROR GOTO Eine Programmverzweigung zu einer wählbaren Programmzeile bei Unterbrechung durch eine Fehlermeldung wird ausgeführt.

ON. . .GOSUB. . . Es wird abhängig vom Wert eines numerischen Ausdrucks zu einer von mehreren angegebenen Subroutinen verzweigt.

ON. . .GOTO. . . Es wird abhängig vom Wert eines numerischen Ausdrucks zu einer von mehreren angegebenen Programmzeilen verzweigt.

ON MENU GOSUB Eine Programmverzweigung zu einer wählbaren Programmzeile bei Menü–Wahl durch den Anwender wird ausgeführt.

ON MOUSE GOSUB Eine Programmverzweigung zu einer wählbaren Programmzeile bei Betätigung der Auswahltaste der Maus wird ausgeführt.

ON TIMER(n) GOSUB Eine Programmverzweigung zu einer wählbaren Programmzeile bei Unterbrechung durch Ablauf einer spezifizierten Zeit wird ausgeführt.

RESTORE Der Lesezeiger der READ–Anweisung wird auf eine spezifizierte DATA–Zeile im Programm gestellt.

Steueranweisungen (Fortsetzung)

RESUME	Das Programm verzweigt nach Ausführung einer Fehlerbearbeitungsroutine zurück ins Hauptprogramm.
RETURN	Das Programm verzweigt nach Ausführung einer Subroutine (ggf. an eine spezifizierte Zeile) ins Hauptprogramm zurück.
SLEEP	Ein Amiga Basic–Programm wird angehalten, bis ein Unterbrechungsereignis eintritt.
STOP	Das Programm wird mit einer Meldung unterbrochen und der Interpreter in die Befehlsebene (Direktmodus) versetzt.
SUB	Benennt ein Amiga Basic–Unterprogramm und leitet es ein.
SWAP	Die Werte zweier Variablen gleichen Typs werden vertauscht.
TIMER ON/OFF/STOP	Die Unterbrechungsreaktionsfähigkeit bei Ablauf einer bestimmten Zeit wird aktiviert oder inaktiviert.
WHILE. . .WEND	Eine Schleife wird solange ausgeführt, wie ein spezifizierter Ausdruck logisch "wahr" (von Null verschieden) ist.

Numerische und logische Funktionen

(liefern numerische oder logisch interpretierbare Werte)

ABS(*x*) Liefert den Absolutwert des Arguments.

ASC(*x$*) Liefert den ASCII–Code des ersten Zeichens des Arguments.

ATN(*x*) Liefert den Arcustangens des im Bogenmaß anzugebenden Arguments.

CDBL(*x*) Liefert den Wert des Arguments als doppeltgenaue Zahl.

CINT(*x*) Liefert eine ganze Zahl durch Runden des Absolutwerts des Arguments.

CLNG(*x*) Liefert eine lange Ganzzahl durch Runden des Absolutwertes des Arguments.

COLLISION(*n*) Liefert kollisionsbezogene Parameter eines grafischen Objektes.

COS(*x*) Liefert den Cosinus des im Bogenmaß anzugebenden Arguments.

CSNG(*x*) Liefert den Wert des Arguments als einfachgenaue Zahl.

CVI(*x$*)
CVL(*x$*)
CVS(*x$*)
CVD(*x$*) Liefert das als Zahl interpretierte Argument in der spezifizierten Genauigkeit

EOF(*f*) Liefert die logische Dateiende-Bedingung (–1 oder 0) für die Datei *f*.

EXP(*x*) Liefert den Exponentialfunktionswert e hoch *x*.

FIX(*x*) Liefert eine ganze Zahl durch Abschneiden der Stellen hinter dem Dezimalpunkt des Arguments.

Numerische und logische Funktionen

(Fortsetzung)

FRE(*x*) Liefert je nach Argument verschiedene Aussagen über verfügbaren Speicherbereich.

INSTR(*n,x$,y$*) Liefert die Position des ersten Auftretens einer Teilzeichenkette in einer Zeichenkette.

INT(*x*) Liefert den größten ganzzahligen Wert, der kleiner oder gleich dem Argument ist.

LBOUND Liefert die untere Grenze der angegebenen Dimension einer Feldvariablen.

LEN(*x$*) Liefert die Länge des Zeichenkettenarguments.

LOC(*n*) Liefert die

- Nummer des letzten gelesenen oder geschriebenen Datensektors bei einer Direktzugriffsdatei;
- Anzahl der aus einer sequentiellen Datei gelesenen Datensektoren.

LOF(*n*) Liefert die Länge einer spezifizierten Datei.

LOG(*x*) Liefert den natürlichen Logarithmus des Arguments.

LPOS(*n*) Liefert die Position des zuletzt auf dem spezifizierten Drucker gedruckten Zeichens im Druckpuffer.

MENU(*n*) Liefert die Nummer des zuletzt gewählten Menu–Titels oder –Punktes.

MOUSE(*n*) Liefert je nach Argument verschiedene Informationen über den Status der Maus–Auswahltaste oder die Position des Mauszeigers innerhalb des aktiven Fensters.

OBJECT.VX/VY Liefert für ein angegebenes grafisches Objekt in Geschwindigkeit in X– bzw. Y–Richtung in Bildpunkten/Sekunde.

Numerische und logische Funktionen

(Fortsetzung)

OBJECT.X/Y	Liefert die Koordinaten der linken oberen Ecke des Rechteckes, in dem das angegebene Objekt definiert ist.
PEEK(*n*)	Liefert das Dezimaläquivalent des binären Inhaltes der Hauptspeicheradresse *n* als Byte–Wert.
PEEKL(*n*)	Liefert das Dezimaläquivalent des binären Inhaltes der Hauptspeicheradresse *n* als lange Ganzzahl.
PEEKW(*n*)	Liefert das Dezimaläquivalent des binären Inhaltes der Hauptspeicheradresse *n* als kurze Ganzzahl.
POINT(*x,y*)	Liefert die Farbkennung eines bestimmten Bildpunktes.
POS(*n*)	Liefert die aktuelle Spaltenposition des Cursors.
RND(*x*)	Liefert eine Zufallszahl.
SADD(*x*)	Liefert die aktuelle Speicheradresse des ersten Zeichens eines angegebenen Zeichenkettenausdrucks.
SGN(*x*)	Liefert das Vorzeichen des Argumentes als numerischen Wert (–1, 0, 1).
SIN(*x*)	Liefert des Sinus des im Bogenmaß anzugebenden Arguments.
SQR(*x*)	Liefert die Quadratwurzel des Arguments.
STICK(*n*)	Liefert die Bildschirmkoordinaten eines wählbaren Joysticks (Spielpult).
STRIG(*n*)	Liefert den Status eines wählbaren Joystick–Knopfes.
TAN(*x*)	Liefert den Tangens des im Bogenmaß anzugebenden Arguments.

Numerische und logische Funktionen

(Fortsetzung)

TIMER	Liefert die Anzahl der Sekunden, die seit Mitternacht oder Systemwiederanlauf vergangen sind.
UBOUND	Liefert die obere Grenze der angegebenen Dimension einer Feldvariablen.
VAL(*x$*)	Liefert das numerische Äquivalent eines Ziffern–Zeichenkettenausdrucks.
VARPTR(*v*)	Liefert die Adresse des ersten Datenbytes einer wählbaren Variablen.
WINDOW(*n*)	Liefert je nach Argument verschiedene Informationen über das aktuelle Ausgabefenster.

Zeichenkettenfunktionen

(liefern Zeichenketten)

CHR$(*n*)	Liefert ein Zeichen mit dem ASCII–Code *n*.
HEX$(*n*)	Liefert die hexadezimale Darstellung des Arguments als Zeichenkette.
INKEY$	Liefert ein Zeichen von der Tastatur.
INPUT$	Liefert eine Zeichenkette wählbarer Länge von der Tastatur oder einer sequentiellen Eingabedatei oder –einheit.
LEFT$(*x$,n*)	Liefert eine Teilzeichenkette aus den ersten *n* Bytes von *x$*.
MID$(*x$,n,m*)	Liefert eine Teilzeichenkette der Länge *m* von der Position *n* an in *x$*.
MKI$(*x*) **MKL$(*x*)** **MKS$(*x*)** **MKD$(*x*)**	Interpretiert je nach Genauigkeit das Argument als unterschiedlich lange Zeichenkette
OCT$(*n*)	Liefert die oktale Darstellung des Arguments als Zeichenkette.
PTAB(*n*)	Die Druckposition wird auf die angegebene Zahl von Bildpunkten vom linken Fensterrand nach rechts gesetzt.
RIGHT$(*x$,n*)	Liefert eine Teilzeichenkette aus den letzten *n* Bytes von *x$*.
SPACE$(*n*)	Liefert eine Zeichenkette aus *n* Leerstellen.
SPC(*n*)	Liefert bei der Ausgabe auf Bildschirm, Drucker oder in sequentielle Datei *n* Leerstellen.
STR$(*n*)	Liefert das Zeichenkettenäquivalent des numerischen Arguments.

Zeichenkettenfunktionen (Fortsetzung)

STRING$ Liefert ein Zeichen eines wählbaren ASCII–Codes oder eine Zeichenkette wählbarer Länge, die aus dem ersten Zeichen einer wählbaren Zeichenkette besteht.

TAB(*n*) Liefert vor der Ausgabe von Daten auf Bildschirm, Drucker oder in sequentielle Datei Leerstellen bis zum Erreichen der durch *n* definierten Druckposition.

TRANSLATE$(*x$*) Erzeugt und übergibt eine Folge von Phonem–Codes für die angloamerikanische Sprachausgabe.

UCASE$(*x$*) Übergibt eine Kopie einer angegebenen Zeichenkette, in der alle Kleinbuchstaben in Großbuchstaben gewandelt sind.

Systemvariablen

CSRLIN Enthält die Zeilenposition des Cursors in der aktuellen Bildschirmseite.

DATE$ Enthält das aktuelle Tagesdatum, das die Systemuhr liefert.

ERL Enthält die Nummer der Programmzeile, in der der Interpreter zuletzt einen Fehler diagnostiziert hat.

ERR Enthält den Code des letzten vom Interpreter diagnostizierten Fehlers.

TIME$ Enthält die aktuelle Zeit, die die Systemuhr liefert.

ABS–Funktion

Format: ***v*=ABS(*x*)**

Bedeutung: Es wird der Absolutwert des numerischen Ausdrucks ***x*** übergeben. Der Absolutwert ist entweder 0 (Null) oder positiv.

Beispiel:

```
Ok
PRINT ABS(2*(5-8))
 6
Ok
```

AREA–Anweisung

Format: **AREA [STEP] (*x,y*)**

Bedeutung: Definiert einen Punkt eines Polygons, das mit der AREAFILL–Anweisung (s. dort) ausgemalt werden kann.

x,y Absolute Bildschirmkoordinaten in Bildpunkten für den Polygon–Eckpunkt. Die maximal erlaubten Koordinatenwerte hängen von der Größe des aktuellen Ausgabe–Fensters ab (s. WINDOW–Anweisung und –Funktion).

STEP Wird STEP angegeben, so sind ***x,y*** zur aktuellen Position des grafischen Cursors relative Koordinaten.

Beispiel: s. AREAFILL–Anweisung.

AREAFILL–Anweisung

Format: **AREAFILL** [***Modus***]

Bedeutung: Malt das durch mehrere AREA–Anweisungen (s. dort) definierte Polygon mit einem wählbaren Muster aus.

Modus Ein ganzahliger Wert von 0 oder 1, der festlegt, wie das Polygon ausgemalt wird.

0 Das Polygon wird mit dem mit der PATTERN–Anweisung (s. dort) festgelegten Muster ausgemalt. Dies ist der voreingestellte Wert. Bei voreingestelltem PATTERN–Wert wird das Polygon weiß ausgemalt.

1 Das Polygon–Innere wird invertiert ausgemalt. Bei voreingestelltem PATTERN–Wert bedeutet dies orange.

Beispiel:

```
AREA (10,10)
AREA STEP (0,40)
AREA STEP (40,-40)
AREAFILL
```

Dies Beispiel zeichnet oben links im Ausgabe–Fenster ein auf der Spitze stehendes, rechtwinkeliges Dreieck, das weiß ausgemalt wird.

ASC–Funktion

Format: ***v*=ASC(*x$*)**

Bedeutung: Es wird der ASCII–Code des ersten Zeichens der Zeichenkette ***x$*** als Dezimalwert übergeben. Der ASCII–Code ist ein numerischer Wert (s. Anhang A). Enthält ***x$*** keine Zeichen (Leerstring), so meldet Amiga Basic den Fehler

```
Illegal function call
```

(unerlaubter Funktionsaufruf).

Beispiel:

```
Ok
X$="Zahl":PRINT ASC(X$)
 90
Ok
```

ATN–Funktion

Format: ***v*=ATN(*x*)**

Bedeutung: Es wird ein Wert im Bogenmaß übergeben, der aus dem numerischen Ausdruck ***x*** errechnet wird und zwischen –PI/2 und + PI/2 liegt. Das Ergebnis ist also der Hauptwert des Arcustangens. Für die Umrechnung in das Gradmaß gilt

WINKEL = 180/PI*ATN(*x*)

Beispiel 1:

```
Ok
PRINT ATN(3)
 1.249046
Ok
```

Beispiel 2:

```
PI=3.141593
BOGEN=ATN(1)
GRAD=BOGEN*180/PI
PRINT BOGEN,GRAD .7853982      45
Ok
```

Anmerkungen:

- Der Hauptwert des Arcustangens wird in einfacher Genauigkeit berechnet, wenn das Argument ebenfalls einfach genau ist und in doppelter Genauigkeit bei doppeltgenauem Argument.

BEEP–Anweisung

Format: **BEEP**

Bedeutung: Es wird für ca. eine Viertelsekunde ein Ton über den Lautsprecher ausgegeben. Zusätzlich blitzt die Bildschirmanzeige kurz auf.

Beispiel:

```
IF FRE(0)<100 THEN
    BEEP
    LOCATE 17,1
    PRINT"Speicherüberlauf: ";
    PRINT"Fenster verkleinern"
END IF
```

Anmerkungen:

- BEEP wirkt genau wie PRINT CHR$(7).

BREAK–Anweisung

Format: **BREAK ON**
BREAK OFF
BREAK STOP

Bedeutung: Die Reaktionsfähigkeit bei Programmunterbrechung durch den Anwender wird aktiviert oder inaktiviert.

Beispiel:

```
BREAK ON
ON BREAK GOSUB Hinweis
DIM PREIS(99),EING(99),AUSG(99),BEST(99)
STUECKZ=2000
OPEN "O",#1,"LAGER"
FOR I=1 TO STUECKZ
    WRITE#1,PREIS(I),EING(I),AUSG(I),BEST(I)
NEXT I
CLOSE #1:BREAK OFF
INPUT "Liste drucken (j/n) ";ANTW$
IF ANTW$="j" THEN BREAK ON:GOSUB DRUCK
END
Hinweis:
    CLS:BEEP
    PRINT"Waehrend des Schreibens keine Unterbre-
    chung möglich"
    RETURN
```

Anmerkungen:

- Die Anweisung BREAK ON muß ausgeführt werden, um eine Unterbrechungsverzweigung mit Hilfe der Anweisung ON BREAK GOSUB (s. dort) zu erreichen, wenn die Tastenkombination Amiga–. gedrückt oder **Stop** aus dem **Run**-Menü gewählt wird.

- Nach BREAK OFF kann bei Programmunterbrechung nicht mehr programmiert verzweigt werden.

- Nach BREAK STOP speichert Amiga Basic eine Programmunterbrechung durch den Anwender, unterbricht das Programm mit einer Verzweigung aber erst, sobald eine BREAK ON–Anweisung gegeben wird.

CALL–Anweisung

Format:

Syntax 1: **[CALL]** ***Sprungmarke*** **[(*Argumentliste*)]**
Syntax 2: **[CALL]** ***num Var*** **[(*Argumentliste*)]**

Bedeutung: Es wird ein Amiga Basic–Unterprogramm, wie mit der SUB–Anweisung (s. dort) definiert (Syntax 1), ein Maschinensprache–Unterprogramm (Syntax 2) oder eine Maschinensprache-Subroutine aus einer Bibliothek, wie mit der LIBRARY–Anweisung (s. dort) definiert (Syntax 2) aufgerufen und es werden wahlweise Parameter übergeben. Das Schlüsselwort CALL kann weggelassen werde. In diesem Fall können auch die Klammern um *Argumentliste* weggelassen werden. Die drei Bedeutungen der CALL–Anweisung werden im folgenden beschrieben:

Aufruf von Amiga Basic–Unterprogrammen

Amiga Basic–Unterprogramme werden mit der SUB–Anweisung (s. dort) benannt und eingeleitet. Es können Variablen oder auch Ausdrücke als Parameter übergeben werden. Variablen werden durch Referenz, Ausdrücke durch ihren Wert übergeben (s. dazu Kapitel 6.1). Z.B.:

```
SUB ALPHA (x,y) STATIC
END SUB
CALL ALPHA (a,b)
```

Näheres zu Amiga Basic–Unterprogrammen und der Parameterübergabe finden Sie in Kapitel 6.1.

Aufruf von Maschinensprache–Unterprogrammen

Die CALL–Anweisung ist die einzige Möglichkeit, die Programmsteuerung an ein **externes** Unterprogramm zu übertragen. In diesem Fall identifiziert der Name eine einfache Variable, deren Wert eine Adresse darstellt, die wiederum den Einsprungpunkt des Unterprogramms im Hauptspeicher angibt. Der Name darf kein Element einer Feldvariablen sein.

Die Argumentliste enthält die Parameter, die an das Unterprogramm übergeben werden sollen. Die Parameter werden gemäß den Aufrufkonventionen der Programmiersprache C übergeben. Sie müssen entweder vom Typ kurze oder lange Ganzzahl sein; andernfalls wird eine Type Mismatch (fehlende Typübereinstimmung)–Fehlermeldung erzeugt. Die Adresse einer Variablen einfacher oder doppelter Genauigkeit kann mit Hilfe der VARPTR–Funktion übergeben werden:

```
CALL Routine(VARPTR(x))
```

Achtung: Es muß sichergestellt sein, daß die Maschinenroutine auf einer **geraden** Speicheradresse beginnt, sonst ist ein Systemabsturz unvermeidlich.

Die Adresse einer Zeichenkette kann mit Hilfe der SADD–Funktion übergeben werden:

```
CALL Routine(SADD(X$))
```

In dem folgenden Beispiel ist die Variable, deren Wert den Einsprungpunkt der Routine darstellt, eine kurze Ganzzahl (&) (Bei 24–Bit–Adressen muß eine lange Ganzzahl–Variable verwendet werden):

```
a=0:b=0
DIM Code%(100)
FOR i=0 TO 90
    READ Code%(i)
NEXT i
CodeAdr&=VARPTR(Code%(0))
CALL CodeAdr&(a,b)
```

Weitere Einzelheiten zum Aufruf von Maschinensprache–Unterprogrammen finden Sie in Kapitel 6.2.

Aufruf von Bibliotheksroutinen

Bibliotheksroutinen sind spezielle Amiga–Programmdateien, die während der Laufzeit dynamisch mit Amiga Basic zusammengebunden werden. Parameter werden entsprechend den Aufrufkonventionen der Sprache C mit ihrem Wert übergeben. Z.B.:

```
LIBRARY "graphics.library"
CALL Draw(50,60)
```

Bei diesem Beispiel wird eine Variable namens **Draw** erzeugt. Dann wird die Information darüber, wo sich die Routine im Speicher befindet, zugewiesen. Aus diesem Grund kann die Variable nicht vom Typ Ganzzahl sein.

So würde die Anweisungsfolge

```
DEFINT A-Z
CALL Draw(50,60)
```

zu der Fehlermeldung **Type Mismatch** (fehlende Typübereinstimmung) führen, wohingegen

```
DEFINT A-Z
Call Draw#(50,60)
```

akzeptiert würde, da die Definition einer doppeltgenauen Gleitpunktvariablen mit Hilfe des #–Zeichens vor der globalen Typdefinition mit DEFINT Priorität hat.

Achtung: Da das Schlüsselwort CALL weggelassen werden kann, kann ein Aufruf auch mit der Syntax

Name Argumentliste

ausgeführt werden. Solch eine CALL–Anweisung kann mit einer alphanumenischen Sprungmarke verwechselt werden. Zur Verdeutlichung:

```
ALPHA: LET A=5
```

Bei diesem Beispiel ist nicht eindeutig klar, ob entweder ein Unterprogramm namens ALPHA ohne Argumentliste aufgerufen werden soll, oder ob der Anweisung LET A=% die alphanumerische Sprungmarke ALPHA: vorangestellt ist. In einem solchen Fall würde Amiga Basic von einer Sprungmarke anstatt von einem argumentlosen Unterprogrammaufruf ausgehen.

Soll ein Unterprogrammaufruf nach der Klausel THEN oder ELSE bei einer IF–Anweisung erfolgen, **muß** das Schlüsselwort CALL angegeben werden.

CDBL–Funktion

Format: ***v*= CDBL(*x*)**

Bedeutung: Der Wert des numerischen Ausdrucks *x* wird in eine doppeltgenaue Zahl gewandelt.

Beispiel:

```
Ok
X=12.289:PRINT X,CDBL(X)
 12.289            12.28899955749512
Ok
```

Der Wert von CDBL(X) ist nach dem Runden nur bis zur dritten Dezimalstelle genau, da X nur mit drei Dezimalstellen definiert wurde. Die restlichen Ziffern sind daher ohne Bedeutung.

Anmerkungen:

- Die Typenwandlung bei numerischen Variablen und die damit verbundenen Rundungsproblematik ist ausführlich in Kapitel 8.5 beschrieben.

CHAIN–Anweisung

Format: **CHAIN [MERGE]** ***Dateiangabe***[,[***Zeile***]
[,[**ALL**][,**DELETE** ***Bereich***]]]

Bedeutung: Es wird ein anderes Amiga Basic–Programm aufgerufen, an das Variable aus dem aufrufenden Programm übergeben werden.

Dateiangabe Zeichenkette in Anführungszeichen ("), wie in Kapitel 5.2 angegeben.

Zeile Konstante oder numerischer Ausdruck zur Angabe einer Zeilennummer des aufgerufenen Programms, ab der dieses gestartet werden soll. Wird ***Zeile*** weggelassen, so wird das aufgerufene Programm am Anfang gestartet. Hier darf **keine** alphanumerische Sprungmarke angegeben werden.

ALL Wird **ALL** angegeben, so werden alle Variablen des aufrufenden an das aufgerufene Programm übergeben. Wird **ALL** nicht angegeben, so muß das aufrufende Programm eine COMMON–Anweisung (s. dort) enthalten, falls Variable übergeben werden sollen.

MERGE Wird **MERGE** angegeben, so wird das aufrufende durch das aufgerufene Programm von der angegebenen Zeile an überlagert. Das aufgerufene Programm muß eine Datei im ASCII–Format (s.a. MERGE–Befehl) sein.

DELETE Diese Erweiterung kann nur in Verbindung mit MERGE angegeben werden und löscht nach der Ausführung der aufgerufenen Überlagerung den nach **DELETE** angegebenen Zeilenbereich.

Bereich Zeilenbereich, der im aufrufenden Programm nach Ablauf der Überlagerung gelöscht werden soll. Es können hier sowohl Zeilennummern als auch alphanumerische Sprungmarken, getrennt durch einen Bindestrich (–), angegeben werden.

Beispiel 1:

```
CHAIN "DF0:PROGR1"
```

Das Programm PROGR1 wird vom internen Diskettenlaufwerk geladen und am Anfang gestartet.

Beispiel 2:

```
CHAIN "DF0:PROGR1",1000
```

Wie Beispiel 1. Die Programmausführung beginnt jedoch erst in Zeile 1000.

Beispiel 3:

```
CHAIN "DF0:PROGR1",1000,ALL
```

Wie Beispiel 2. Es werden jedoch alle Variablen des aufrufenden Programms an PROGR1 übergeben.

Beispiel 4:

```
CHAIN MERGE "DF0:PROGR2",100
```

Das aufrufende Programm wird durch PROGR2 überlagert. Die Überlagerung wird mit Zeile 100 gestartet.

Beispiel 5:

```
CHAIN MERGE "DF0:PROGR2",100,DELETE 100-3000
```

Wie Beispiel 4. Nach der Ausführung der Überlagerung PROGR2 wird diese jedoch von Zeile 100 bis 3000 gelöscht.

Anmerkungen:

- Bereits im aufrufenden Programm eröffnete Dateien bleiben nach Ausführung der CHAIN–Anweisung geöffnet.
- Ein mit OPTION BASE (s. dort) gesetzter Wert bleibt nach Ausführung einer CHAIN MERGE–Anweisung erhalten.
- Mit DEF FN oder DEFTyp umdefinierte Variablen müssen bei CHAIN MERGE im aufgerufenen Programm auf ihren alten Status aus dem aufrufenden Programm gesetzt werden, falls gemeinsame Variablen oder definierte Funktionen verwendet werden.
- Vor der Ausführung eines mit CHAIN geladenen Programms wird die RESTORE–Anweisung (s. dort) automatisch ausgeführt, so daß die nächste READ–Anweisung (s. dort) auf die erste DATA–Zeile des nach der CHAIN–Anweisung im Hauptspeicher befindlichen Amiga–Basic–Programms zugreift.
- Wird MERGE zusammen mit DELETE verwendet, so sollten anwenderspezifische Funktionen vor dem zu löschenden Programmbereich definiert werden. Andernfalls sind sie nach dem Löschen undefiniert.

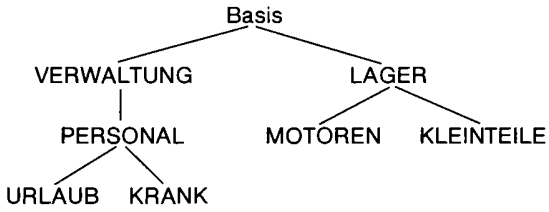
CHDIR–Befehl

Format: **CHDIR** ***Pfad***

Bedeutung: Ein anderes als das aktuelle Disk– (Disketten– oder Festplatten–) Verzeichnis wird aktualisiert (s.a. Kapitel 1.3.3 im AmigaDOS–Benutzerhandbuch).

Pfad Eine Zeichenkette in Anführungszeichen ("), die angibt, wie ein in dieser Zeichenkette spezifiziertes Verzeichnis erreicht wird (s.a. Kapitel 5.2).

Beispiele: Bei den folgenden Beispielen wird von der unten dargestellten Verzeichnisstruktur ausgegangen:



Beispiel 1:
```
CHDIR ":"
```

Das Basis–Verzeichnis (root) wird aktuelles Disk–Verzeichnis.

Beispiel 2:
```
CHDIR "VERWALTUNG/PERSONAL/URLAUB"
```

Das Verzeichnis URLAUB wird anstelle des Basis–Verzeichnisses aktuelles Verzeichnis.

Beispiel 3:
```
CHDIR "KLEINTEILE"
```

Das Verzeichnis "KLEINTEILE" wird anstelle des Verzeichnisses LAGER zum aktuellen Verzeichnis.

Beispiel 4:

```
CHDIR "/"
```

Das Verzeichnis VERWALTUNG wird aktuelles Verzeichnis, falls PERSONAL vorher aktuelles Verzeichnis war. Ein Verzeichnisname in Form eines Schrägstriches ("/") kennzeichnet immer das hierarchisch nächsthöhere Verzeichnis.

Beispiel 5:

```
CHDIR "VERWALTUNG"
CHDIR "DF1:LAGER"
```

Das Verzeichnis VERWALTUNG wird anstelle des Basis–Verzeichnisses zum aktuellen Verzeichnis auf dem aktuellen Disk-Laufwerk (hier DF0:). LAGER wird zum aktuellen Verzeichnis auf Disk–Laufwerk DF1:, auf dem es auch vorhanden sein muß. Dateiangaben für diese Laufwerke beziehen sich dann automatisch auf in diesen Verzeichnissen enthaltene Dateien.

CHR$–Funktion

Format: ***v$*** **= CHR$(*n*)**

Bedeutung: Es wird eine Ein–Zeichenkette übergeben, die aus dem Zeichen besteht, dessen ASCII–Code (s. Anhang A) durch den Wert des ganzzahligen Ausdrucks ***n*** gegeben ist.

n Ein ganzzahliger Ausdruck im Bereich zwischen 0 und 255.

Beispiele:

```
PRINT CHR$(7)
```

Es wird ein kurzer Ton über den Lautsprecher ausgegeben (s.a. BEEP–Anweisung).

```
PRINT CHR$(12)
```

Das Ausgabefenster wird gelöscht und der Cursor wird in die linke obere Fensterecke (Home–Position) gesetzt.

```
CLS
FOR I=65 TO 90
    PRINT CHR$(I)" ";
NEXT
```

Es werden die Großbuchstaben des Alphabets angezeigt.

CINT–Funktion

Format: ***v*= CINT(*x*)**

Bedeutung: Es wird eine ganze Zahl übergeben, die sich durch Runden des Absolutwertes des numerischen Ausdrucks *x* ergibt.

x Beliebiger numerischer Ausdruck, dessen Wert zwischen –32768 und 32767 liegen muß.

Beispiel 1:

```
Ok
PRINT CINT(12.51)
 13
Ok
```

Beispiel 2:

```
Ok
PRINT CINT(12.5)CINT(13.5)
 12  14
Ok
```

Beispiel 3:

```
Ok
PRINT CINT(-14.8)
-15
Ok
```

Anmerkungen:

- Liegt der Wert von *x* außerhalb des erlaubten Bereiches, wird der Fehler

  ```
  Overflow
  ```

  (Überlauf) angezeigt.

- Ist der ganzzahlige Teil des numerischen Ausdruckes gerade, so rundet CINT ab, ist er ungerade, so rundet CINT auf (s. Beispiel 2).

- Siehe auch bei den Funktionen FIX und INT.

CIRCLE–Anweisung

Format: **CIRCLE [STEP] (*x,y*),*r*[,*Farbe*[,*Start,Ende*[,*Bild*]]]**

Bedeutung: Es wird eine Ellipse mit dem Mittelpunkt **(*x,y*)** und dem Radius ***r*** auf dem Bildschirm gezeichnet.

x,y Die absoluten oder relativen (mit STEP) Bildschirmkoordinaten des Mittelpunktes der Ellipse.

r Ein ganzzahliger Ausdruck, dessen Wert den Radius (Hauptachse), gemessen in Bildpunkten in horizontaler Richtung, angibt.

Farbe Eine Farbnummer zwischen 0 und 3 entsprechend der mit einer PALETTE–Anweisung (s. dort) gesetzten Farbkennung. Voreingestellt ist die Vordergrundfarbe, die mit der COLOR-Anweisung (s. dort) eingestellt wurde.

Start,Ende Numerische Ausdrücke, deren Werte Winkelangaben im Bogenmaß darstellen und die angeben, wo das Zeichnen der Ellipse beginnen und enden soll. Die Winkel werden getreu der mathematischen Konvention von 0 nach links gezählt und können im Bereich zwischen –2*PI und 2*PI angegeben werden.

Bild Ein numerischer Ausdruck, dessen Wert das Verhältnis von Breite zu Höhe eines Bildpunktes (Bildverhältnis) definiert. Das Bildverhältnis ist bei unterschiedlichen Monitortypen verschieden. Wenn in der CIRCLE–Anweisung hier das Bildverhältnis des verwendeten Monitors angegeben wird, wird ein exakter Kreis, andernfalls eines Ellipse gzeichnet.

Das Bildverhältnis für den Standard–Amiga–Monitor bei hoher Auflösung im 640x200–Bildschirm beträgt 2.25:1 oder etwa 0.44. Dieser Wert ist für ***Bild*** voreingestellt.

Beispiel:

```
BVERH=.1
WHILE BVERH<20
    CIRCLE (60,60),55,1,,,BVERH
    BVERH=BVERH*1.4
WEND
```

Dieses Beispiel zeichnet um einen Mittelpunkt eine Reihe von weißen (falls nicht durch PALETTE–Anweisung geändert) Ellipsen, deren Haupt– und Nebenachsen wegen des variierten Bildverhältnisses ebenfalls variieren.

CLEAR–Befehl

Format: **CLEAR** [,[*Prg*][,*Stap*]]

Bedeutung: Die Inhalte aller Variablen werden durch Nullsetzen oder Zuweisen von Zeichenketten der Länge Null (Leerstrings) gelöscht. Ggf. wird die Obergrenze des für den Interpreter verfügbaren Programm– und Stapelspeichers definiert.

Prg Numerischer Ausdruck, dessen Wert zwischen 1024 und dem Oberwert des verfügbaren Systemspeichers liegen muß und der den für den Interpreter verfügbaren Basic–Programmspeicher dimensioniert.

Stap Numerischer Ausdruck, dessen Wert zwischen 1024 und dem Oberwert des verfügbaren Systemspeichers liegen muß und der den für den Interpreter verfügbaren Basic–Stapelspeicher dimensioniert.

Beispiel 1:

```
CLEAR
```

Die Inhalte aller Variablen werden gelöscht.

Beispiel 2:

```
CLEAR ,40000
```

Wie Beispiel 1. Außerdem werden dem Interpreter 40000 Bytes als verfügbarer Programmspeicher reserviert.

Beispiel 3:

```
CLEAR ,,1024
```

Wie Beispiel 1. Außerdem werden dem Interpreter 1 KBytes als Stapelspeicher reserviert.

Beispiel 4:

```
CLEAR ,40000,1024
```

Faßt Beispiele 2 und 3 zusammen.

Anmerkungen:

- Voreingestellte Werte für ***Prg*** und ***Stap*** sind 25000 bzw. 4789
- ***Prg*** wird heraufgesetzt, wenn für Amiga Basic ein größeres Datensegment benötigt wird.
- ***Stap*** wird heraufgesetzt, wenn das Programm viele tiefgeschachtelte Schleifen (FOR. . .NEXT) oder Subroutinenaufrufe (GOSUB) enthält.
- Neben allen Variablen werden durch CLEAR alle mit DEF-Anweisungen vereinbarten Definitionen zurückgesetzt.
- Alle geöffneten Dateien werden geschlossen, und zugewiesene Puffer werden freigegeben.
- Siehe auch FRE–Funktion.

CLNG–Funktion

Format: ***v*=CLNG(*x*)**

Bedeutung: Es wird eine lange Ganzzahl übergeben, die sich durch Runden des Absolutwertes des numerischen Ausdrucks *x* ergibt.

x Beliebiger numerischer Ausdruck, dessen Wert zwischen –2147483648 und 2147483647 liegen muß.

Beispiel 1:

```
Ok
PRINT CLNG(33255.56)
33256
Ok
```

Beispiel 2:

```
Ok
PRINT CLNG(18051964.5)CLNG(12061347.85)
18051964 12061348
Ok
```

Beispiel 3:

```
Ok
PRINT CLNG(-34589.7)
-34590
Ok
```

Anmerkungen:

- Liegt der Wert von *x* außerhalb des erlaubten Bereiches, wird der Fehler

  ```
  Overflow
  ```

  (Überlauf) angezeigt.

- Ist der ganzzahlige Teil des numerischen Ausdruckes gerade, so rundet CLNG ab, ist er ungerade, so rundet CLNG auf (s. Beispiel 2).

CLOSE–Anweisung

Format: **CLOSE** [[#]***Dateinr***[,[#]***Dateinr***]. . .]

Bedeutung: Die Ein– oder Ausgabe von bzw. an Ein–/Ausgabegeräte wird beendet.

Dateinr Die in der OPEN–Anweisung (s. dort) angegebene logische Dateinummer.

Beispiel 1:

```
CLOSE #1,#2,#3
```

Die Ein–/Ausgabe von bzw. an die Geräte oder Dateien, für die ein Ein– oder Ausgabekanal und damit –Puffer unter den Nummern 1, 2 und 3 eingerichtet wurde, wird beendet, und die ggf. zugehörigen Dateien werden geschlossen.

Beispiel 2:

```
CLOSE
```

Alle Ein–/Ausgabedateien oder –kanäle werden geschlossen.

Anmerkungen:

- Die CLOSE–Anweisung übergibt an die spezifizierten Dateien oder Geräte für sequentielle Ausgabe vor dem Schließen der Datei den Inhalt des letzten Datenpuffers.
- Die einer Datei oder einem Ausgabekanal zugeordnete Dateinummer wird durch CLOSE wieder frei und kann für eine neue OPEN–Anweisung wiederverwendet werden.
- Die Anweisungen END, NEW, SYSTEM und CLEAR führen für alle geöffneten Dateien oder Geräte automatisch CLOSE aus.
- Die Anweisung STOP beinhaltet kein CLOSE.
- Siehe auch CLEAR, END, NEW, OPEN, STOP, SYSTEM.

CLS–Anweisung

Format: **CLS**

Bedeutung: Das aktuelle Ausgabefenster wird gelöscht und der Cursore wird in die obere linke Fensterposition gesetzt.

Anmerkungen:

- Es wird nur das **aktuelle** und kein anderes Ausgabefenster durch CLS gelöscht.

COLLISION–Anweisung

Format: **COLLISION ON**
COLLISION OFF
COLLISION STOP

Bedeutung: Die Unterbrechungsreaktionsfähigkeit für Objekt–Kollisionen untereinander oder mit Fensterbegrenzungen wird aktiviert oder inaktiviert.

Beispiel: Siehe OBJECT.SHAPE–Anweisung für ein ausführliches Beispiel.

Anmerkungen:

- COLLISION ON aktiviert die Unterbrechungsreaktionsfähigkeit für Objekt–Kollisionen. In einem Programm prüft der Interpreter dann vor jeder neuen Anweisung, ob eine Objekt–Kollison stattgefunden hat und verzweigt ggf. zu der in der ON COLLISION–Anweisung (s. dort) angegebenen Subroutine.

- Eine COLLISION–Anweisung darf nicht vor einer ON COLLISION–Anweisung im Programm stehen.

- COLLISION OFF inaktiviert die Unterbrechungsreaktionsfähigkeit. Unterbrechungen durch Objekt–Kollisionen sind nach dieser Anweisung dann nicht mehr möglich.

- COLLSISION STOP inaktiviert zunächst auch die Unterbrechungsreaktionsfähigkeit. Die Unterbrechung bei einer Objekt–Kollision erfolgt jedoch sofort, sobald die Anweisung COLLISION ON ausgeführt wird.

- S.a. COLLISION–Funktion sowie Kapitel 6.4 "Verarbeitung von Unterbrechungsereignissen".

COLLISION–Funktion

Format: ***v*= COLLISION(*n*)**

Bedeutung: Liefert abhängig von der Objekt–Kennung ***n*** einen Parameter, der die Aussage über verschiedene Parameter eines beweglichen Objektes (Sprite oder Bob) erlaubt.

n Ein ganzzahliger Ausdruck, dessen Wert entweder eine Objekt–Kennung angibt, die mit der Objekt–Kennung in einer OBJECT.SHAPE–Anweisung (s. dort) korrespondiert (***n*** > 0) oder der die Arbeitsweise der COLLISION–Funktion steuert (***n*** = 0 oder ***n*** = –1):

n = 0
COLLISION(0) liefert die Kennummer des Objektes, das zuletzt mit einem anderen Objekt kollidiert ist. Der Inhalt der Kollisions–Warteschlange, in der alle Kollisionsereignisse zeitlich aufeinanderfolgend gespeichert sind, wird dabei nicht verändert.

n = –1
COLLISION(–1) liefert die Kennummer des Fensters, in dem die durch COLLISION(0) identifizierte Kollision stattgefunden hat.

n < > 0
Mit einem Parameter ungleich Null liefert diese Funktion die Kennung des Objektes, das mit dem Objekt ***n*** kollidiert ist und entfernt gleichzeitig die Information aus der Kollisionswarteschlange.

Liefert die COLLISION–Funktion einen Wert zwischen –1 und –4, so bedeutet dies, daß Objekt ***n*** mit einer der vier Fensterbegrenzungen kollidiert ist:

–1 obere Begrenzung

–2 linke Begrenzung

–3 untere Begrenzung

–4 rechte Begrenzung

Beispiel: Siehe OBJECT.SHAPE–Anweisung für ein ausführliches Beispiel.

COLOR–Anweisung

Format: **COLOR** [***Vordergrund***][,***Hintergrund***]

Bedeutung: Setzt die Vordergrund– und Hintergrundfarben, die Amiga Basic für Text– und grafische Anzeigen auf dem Bildschirm benutzt.

Vordergrund Ein ganzzahliger Ausdruck, dessen Wert zwischen 0 und 3 liegen muß, und der der Farbkennung in der letzten PALETTE–Anweisung (s. dort) entspricht. Dieser Wert legt die Vordergrundfarbe für die Anzeige von Text oder grafischen Objekten fest.

Hintergrund Ein ganzzahliger Ausdruck, dessen Wert zwischen 0 und 3 liegen muß, und der der Farbkennung in der letzten PALETTE–Anweisung (s. dort) entspricht. Dieser Wert legt die Hintergrundfarbe fest, auf der Text oder grafische Objekte angezeigt werden.

Wird einer der Parameter oder werden beide weggelassen, so werden, falls nicht anders durch eine PALETTE–Anweisung festgelegt, die voreingestellten Systemfarben weiß für den Vordergrund und blau für den Hintergrund gesetzt.

Beispiel:

```
PALETTE 1,RND,RND,RND
PALETTE 2,RND,RND,RND
COLOR 1,2
```

Bei diesem Beispiel werden für Vordergrund– und Hintergrund zufällige Farbmischungen erzeugt, die dann gesetzt werden.

Anmerkungen:

- Amiga Basic verwendet generell die Standard–Systemfarben weiß für Vordergrund und blau für Hintergrund, falls nichts anderes mit der PALETTE–Anweisung (s. dort) festgelegt wird. Die Systemfarben können außerdem vom Anwender mit dem Voreinsteller–Programm Pereferences (s. Amiga–Benutzerhandbuch) festgelegt werden.

COMMON–Anweisung

Format: **COMMON** ***Variable***[,***Variable***]. . .

Bedeutung: Die in der Anweisung angegebenen Variablen werden an ein mit der CHAIN–Anweisung (s. dort) aufgerufenes Amiga Basic–Programm übergeben.

Variable Name der zu übergebenden Variablen. Feldvariablen werden durch () hinter dem Namen gekennzeichnet.

Beispiel:

```
COMMON X,Y,Z(),X$,Y$()
CHAIN "DF1:PROG1"
```

Das Programm PROG1 wird von Laufwerk DF1: geladen und bekommt vom aufrufenden Programm die Variablen X,Y, und X$ sowie die Feldvariablen Z und Y$ übergeben.

Anmerkungen:

- Übergebene Feldvariablen brauchen im aufgerufenen Programm nicht dimensioniert zu werden.
- Es sind beliebig viele COMMON–Anweisungen im aufrufenden Programm erlaubt, die an beliebiger Stelle stehen dürfen. Es empfiehlt sich jedoch, COMMON–Anweisungen an den Programmanfang zu stellen, um zu verhindern, daß die Programmsteuerung an ein anderes Programm übergeben wird, bevor durch Ausführen der COMMON–Anweisung die zu übergebenden Variablen definiert wurden.
- Ein Variablenname darf nicht mehrmals in mehreren COMMON–Anweisungen vorkommen.
- Sollen alle Variablen übergeben werden, so wird COMMON weggelassen und die CHAIN–Anweisung mit dem Zusatz ALL verwendet.

CONT–Befehl

Format: **CONT**

Bedeutung: Ein durch einen Fehler, durch gleichzeitiges Drücken der Tasten AMIGA–. oder durch die Anweisungen STOP oder END abgebrochenes bzw. beendetes Programm wird am Unterbrechungs– oder Beendigungspunkt fortgesetzt, es sei denn, es wurde während der Unterbrechung verändert.

Beispiel:

```
INPUT A,B,C
K=A*5:L=B*3:PRINT L
STOP
M=C*K+100:PRINT M
```

Wird dieses Beispiel mit RUN gestartet, so ergibt sich im Ausgabefenster folgende Anzeige, wenn drei Werte und zwischendurch CONT eingegeben werden:

```
? 1,2,3
 6
Ok
cont
 115
Ok
```

Anmerkungen:

- Wird das Programm bei der Aufforderung zur Dateneingabe mit der INPUT– oder der LINE INPUT–Anweisung abgebrochen, so setzt CONT das Programm mit der Wiederholung dieser Anweisung fort. In allen anderen Fällen wird mit der auf die abgebrochene Anweisung folgenden Anweisung fortgesetzt.

- CONT in Verbindung mit STOP erlaubt das schrittweise Austesten von Programmen (s.a. TRON– und TROFF–Befehle).

COS–Funktion

Format: ***v*= COS(*x*)**

Bedeutung: Es wird der Cosinus des Wertes des numerischen Ausdruckes ***x*** berechnet und übergeben. Ist das Argument einfach genau, erfolgt die Berechnung einfach–genau, ist es doppelt–genau, so wird auch ein doppelt–genauer Wert übergeben.

x Winkel im Bogenmaß.

Beispiel:

```
PI=3.141593
WINKEL=180
RADIANS=WINKEL*PI/180
PRINT COS(RADIANS)
```

Wird dieses Programm gestartet so ergibt sich die Anzeige

```
-1
Ok
```

Um den Cosinus des Winkels von 180 Grad zu ermitteln, wird der Winkel in der dritten Programmzeile ins Bogenmaß umgerechnet.

CSNG–Funktion

Format: ***v*= CSNG(*x*)**

Bedeutung: Der Wert des numerischen Ausdrucks ***x*** wird in eine einfachgenaue Zahl umgewandelt.

Beispiel:

```
Ok
X#=12.34210014343262:PRINT X#,CSNG(X#)
 12.34210014343262
12.3421
Ok
```

Anmerkungen:

- Die Typenwandlung bei numerischen Variablen und die damit verbundene Rundungsproblematik ist ausführlich in Kapitel 8.5 beschrieben.
- S.a. CDBL– und CINT–Funktionen.

CSRLIN–Systemvariable

Format: ***v*= CSRLIN**

Bedeutung: Es wird die Nummer der Bildschirmzeile übergeben, in der der Cursor in dem aktuellen Ausgabefenster steht. Die Nummer ist immer größer oder gleich 1.

Beispiel:

```
X=POS(0):Y=CSRLIN
LOCATE 12,38:PRINT "Hallo !"
LOCATE Y,X
```

Nachdem zunächst die aktuelle Cursor–Position zwischengespeichert wurde, wird der Text Hallo! in Schirmzeile 12 ab Spalte 38 im aktuellen Ausgabefenster geschrieben und dann die alte Cursorposition wiederhergestellt.

Anmerkungen:

- Weitere Cursor–bezogene Anweisungen oder Funktionen sind LOCATE und POS (s. dort).

CVI–, CVL–, CVS–, CVD–Funktionen

Formate:

***v*= CVI(*2–Byte–Zeichenkette*)**
***v*= CVS(*4–Byte–Zeichenkette*)**
***v*= CVL(*4–Byte–Zeichenkette*)**
***v*= CVD(*8–Byte–Zeichenkette*)**

Bedeutung:

Der Inhalt von ***Zeichenkette*** wird als Zahl in interner Darstellung interpretiert und in Form eines numerischen Wertes übergeben.
CVI übergibt eine 2–Byte–Zeichenkette als kurze Ganzzahl.
CVL übergibt eine 4–Byte–Zeichenkette als lange Ganzzahl.
CVS übergibt eine 4–Byte–Zeichenkette als einfachgenaue Zahl.
CVD übergibt eine 8–Byte–Zeichenkette als doppeltgenaue Zahl.

Beispiel:

```
FIELD #2,8 AS X$,10 AS Y$
GET #2
Z#=CVD(X$)
```

Aus einer Datei für wahlfreien Zugriff mit in der ersten Zeile definierten Datenfeldern wird in der zweiten Zeile der nächste Datensatz gelesen. In der letzten Zeile des Beispiels werden die ersten 8 Bytes als Zahl doppelter Genauigkeit interpretiert und der Variablen Z# zugewiesen.

Anmerkungen:

- Numerische Werte werden in Dateien für wahlfreien Zugriff grundsätzlich als Zeichenketten (s. MKI$–, MKL$–, MKS$–, MKD$–Funktionen) gespeichert und müssen nach dem Auslesen als numerische Werte zurückgegeben werden.

- Der Unterschied zur VAL–Funktion (s. dort) besteht darin, daß die aktuellen Datenbytes nicht verändert werden. Sie werden nur anders interpretiert.

DATA–Anweisung

Format: **DATA** ***Konstante***[,***Konstante***]. . .

Bedeutung: Es werden Konstanten (Zahlen oder Zeichenketten) im Speicher abgelegt, die mit der READ–Anweisung (s. dort) im Programm gelesen werden können. Es dürfen keine Ausdrücke oder Variablen angegeben werden.

Konstante Jede numerische oder Zeichenkettenkonstante ist erlaubt. Bei numerischen Konstanten ist jeder Typ (s. Kapitel 8.4) erlaubt. Enthalten Zeichenkettenkonstanten Kommata (,), Doppelpunkte (:) oder signifikante führende oder nachfolgende Leerstellen, so müssen sie in Anführungszeichen (") eingeschlossen werden.

Beispiel:

```
DATA "Das ","Wetter ","ist ","schön!"
FOR I=1 to 4
    READ A$:PRINT A$;
NEXT:PRINT
```

Wird dieses Programm gestartet, so ergibt sich folgende Anzeige im Ausgabefenster:

```
Das Wetter ist schön!
Ok
```

Anmerkungen:

- Die Anzahl der DATA–Anweisungen in einem Programm ist nur durch den verfügbaren Speicher begrenzt.
- Die DATA–Anweisung ist eine nichtausführbare Anweisung.
- Eine DATA–Anweisung kann so viele Konstanten enthalten, wie in eine logische Programmzeile passen.
- Alle DATA–Anweisungen eines Programms stellen eine kontinuierliche Liste von Daten dar, unabhängig davon, wieviele andere Programmzeilen zwischen einzelnen DATA–Zeilen liegen.

- Der in der READ–Anweisung deklarierte Variablentyp muß mit dem zu lesenden Konstantentyp übereinstimmen, sonst wird die Fehlermeldung

  Syntax error

  (Syntaxfehler) angezeigt.

- Einzelne DATA–Zeilen können mit der RESTORE–Anweisung (s. dort) wiederholt gelesen werden.

- Mit der READ–Anweisung werden die DATA–Zeilen in aufsteigender Zeilennummernfolge gelesen.

DATE$–Systemvariable

Format: ***v$*=DATE$**

Bedeutung: Es wird eine 10 Zeichen lange Zeichenkette des Formats ***mm–tt–jjjj*** als Datum einer Zeichenkettenvariablen zugeordnet. Das Datum kann vorher mit Hilfe des Voreinsteller–Programmes Preferences (s. Amiga–Benutzerhandbuch) gesetzt werden.

Beispiel:

```
Ok
PRINT DATE$
03-05-1986
```

Anmerkungen:

- Das Datum wird grundsätzlich in der angloamerikanischen Schreibweise (erst der Monat, dann der Tag, dann das Jahr) übergeben.

DECLARE FUNCTION–Anweisung

Format: **DECLARE FUNCTION *Name* [(*Parameterliste*)] LIBRARY**

Bedeutung: Deklariert eine Maschinensprache–Routine aus einer Bibliothek als eine aufrufbare Funktion und definiert ggf. den Typ des Wertes, den die Funktion zurückgibt.

Name ist jeder gültige Name für eine numerische Amiga Basic-Variable, der optional eine der Kennungen %, &, ! oder # angefügt werden kann. Dieser Name bezeichnet die Maschinenspracheroutine und den Typ des Wertes, der ggf. von dieser zurückgegeben wird.

Parameterliste ist die Liste der Parameter, die die Routine bei ihrem Aufruf erwartet. Diese Liste wird von Amiga Basic ignoriert. Für dokumentarische Zwecke sollte sie jedoch stets angegeben werden.

Beispiel:

```
DECLARE FUNCTION ViewPortAddress&() LIBRARY
LIBRARY "intuition.library"
VPA&=ViewPortAddress&(WINDOW(7))
```

Bei diesem Beispiel wird der Variaben VPA& der Wert zugewiesen, den die Bibliotheksroutine ViewPortAddress& zurückgibt.

Anmerkungen:

- Wenn die Bibliotheksfunktion aufgerufen wird (s. CALL-Anweisung), übergibt Amiga Basic alle Parameter, falls vorhanden. Die ggf. dem Funktionsnamen in der DECLARE FUNCTION–Anweisung angehängte Kennung bestimmt den Typ des zurückgegebenen Wertes. Ist keine Kennung angefügt, werden die Standardregeln für die Typkennung angewendet (s. DEF*Typ*–Anweisungen).

- Bei der CALL–Anweisung werden detailliert die Parameterübergabe–Konventionen beschrieben.

- S.a. CSNG, DEFINT, DEFSNG, LIBRARY, CALL.

DEF FN–Anweisung

Format: **DEF FN*Name*[(*Arg*[,*Arg*]. . .)] = *Funktionsdefinition***

Bedeutung: Es wird eine vom Anwender festgelegte Funktion definiert und benannt.

Name Ein gültiger Variablenname, der als Funktionsname dient und ohne Leerstelle hinter FN angegeben werden muß.

Arg Ein Variablenname, der beim Aufruf der Funktion durch die dann aktuelle Variable oder den aktuellen Wert ersetzt wird.

Funktionsdefinition Die Rechenvorschrift zur Berechnung des Funktionswertes. Die Funktionsdefinition darf ein numerischer oder Zeichenkettenausdruck sein, muß aber im Typ mit dem Typ von ***Name*** übereinstimmen.

Beispiel:

```
DEF FNRE(X,Y)=X*Y
PRINT "Rechteckfläche ist"FNRE(5,4)
```

Wird dieses Beispiel mit RUN gestartet, so ergibt dies im Ausgabefenster:

```
Rechteckfläche ist 20
Ok
```

In diesem Beispiel wird eine Funktion zur Rechteckflächenberechnung mit zwei Argumenten definiert.

Anmerkungen:

- In einer Funktionsdefinition kann nur eine Rechenvorschrift angegeben werden.
- Die angegebenen Argumente dienen nur als Platzhalter und haben keinen Einfluß auf Programmvariable gleichen Namens.
- Ist ein in der Rechenvorschrift angegebener Variablenname auch Bestandteil der Argumentliste, so wird beim Funktionsaufruf der Argumentwert mitgegeben; andernfalls der aktuelle Wert der Variablen.

- Bei numerischen Funktionen wird der Wert des Ausdrucks in die durch den Funktionsnamen festgelegte Genauigkeit gewandelt, ehe er übergeben wird.

- Bei fehlender Typübereinstimmung zwischen ***Name*** und ***Funktionsdefinition*** wird die Fehlermeldung

  ```
  Type mismatch
  ```

  (keine Typübereinstimmung) angezeigt.

- Beim Aufruf einer Funktion vor ihrer Definition wird die Fehlermeldung

  ```
  Undefined user function
  ```

  (undefinierte Benutzerfunktion) angezeigt.

- Sich selbst aufrufende (rekursive) Funktionen führen bei endloser (fehlerhafter) Rekursion zur Fehlermeldung

  ```
  Out of memory
  ```

  (Hauptspeicher nicht ausreichend).

- Wenn mehrere Funktionsdefinitionen unter demselben Namen angegeben werden, wird die letzte Definition als die aktuelle angesehen.

DEF*Typ*–Anweisungen

Format: **DEF*Typ* *Buchst*[–*Buchst*][,*Buchst*[–*Buchst*]]...**

Bedeutung: Es werden Variablentypen als

- kurze Ganzzahl
- lange Ganzzahl
- mit einfacher Genauigkeit
- mit doppelter Genauigkeit
- Zeichenkette

definiert.

Typ	**Kennung**
INT = kurze Ganzzahl	%
LNG = lange Ganzzahl	&
SNG = einfache Genauigkeit	!
DBL = doppelte Genauigkeit	#
STR = Zeichenkette	$

Buchst Ein Buchstabe des Alphabets.

Beispiele:

```
DEFINT A-I
```

Alle Variablen, deren Namen mit den Buchstaben A bis I beginnen, werden als kurze Ganzzahlvariablen interpretiert.

```
DEFDBL X
```

Alle Variablen, deren Namen mit X beginnen, werden als numerische Gleitpunktvariablen doppelter Genauigkeit interpretiert.

```
DEFSTR B,X-Z
```

Alle Variablen, deren Namen mit B, X, Y oder Z beginnen, werden als Zeichenkettenvariablen interpretiert.

Anmerkungen:

- Wird eine Variable durch eines der Typenkennzeichen %, &, !, #, oder $ gekennzeichnet, so hat diese Kennzeichnung höhere Priorität als die DEF*Typ*–Anweisung.

- DEF*Typ*–Anweisungen müssen, wenn sie benutzt werden, immer **vor** der ersten Verwendung einer damit definierten Variablen im Programm stehen.

- Fehlen DEF*Typ*–Anweisungen für numerische Variablen in einem Programm, so betrachtet der Interpreter Variablen ohne Typkennzeichnung als numerische Variablen einfacher Genauigkeit.

- DEF*Typ*–Deklarationen gelten grundsätzlich nur für das Programm, in dem sie stehen. Bei Verwendung von CHAIN (s. dort) müssen sie ggf. erneut gesetzt werden.

DELETE–Befehl

Format: **DELETE** [***Marke1***][–[***Marke2***]]

Bedeutung: Der angegebene Zeilennummernbereich wird im Programm gelöscht und Amiga Basic kehrt auf die Befehlsebene zurück.

Marke1 Nummer oder alphanumerische Sprungmarke der ersten zu löschenden Zeile.

Marke2 Nummer oder alphanumerische Sprungmarke der letzten zu löschenden Zeile.

Beispiele:

```
DELETE 10
```

Zeile 10 wird gelöscht.

```
DELETE 10-200
```

Zeilen 10 bis 200 werden gelöscht.

```
DELETE -200
```

Alle Zeilen vom Programmanfang bis Zeile 200 einschließlich werden gelöscht.

```
DELETE Druck-
```

Alle Zeilen von einschließlich Druck: bis zum Programmende werden gelöscht.

Anmerkungen:

- Angabe einer nicht existierenden Programmzeile führt zu der Fehlermeldung

  ```
  Illegal function call
  ```

  (unerlaubter Funktionsaufruf)

DIM–Anweisung

Format: **DIM [SHARED]** ***Var(Ind)*****[,*****Var(Ind)*****]**

Bedeutung: Es wird die maximale Anzahl von Elementen für Feldvariablen beliebigen Typs definiert und entsprechender Speicherplatz reserviert.

Var Name der Feldvariablen.

Ind Eine Liste numerischer Ausdrücke, deren Werte, durch Kommata getrennt, die Anzahl der Elemente je Dimension angeben.

Mit dem Zusatz **SHARED** werden die Variablen als globale Felder dimensioniert, die damit dem Hauptprogramm und allen Unterprogrammen gleichermaßen zur Verfügung stehen. DIM SHARED darf nur im Hauptprogramm angegeben werden.

Durch DIM werden alle Elemente bei numerischen Feldern auf Null und bei Zeichenkettenfeldern auf Länge Null (Leer–Zeichenkette) gesetzt.

Beispiel 1:

```
DIM A$(20),B(8,5)
FOR I=0 TO 20:READ A$(I):NEXT
FOR J=0 TO 8:FOR K=0 TO 5
READ B(J,K):NEXT K,J
```

Zunächst werden ein Zeichenkettenfeld mit 21 sowie ein numerisches, zweidimensionales Feld aus 9 Zeilen und 6 Spalten, also 54 Elementen definiert, die dann in Leseschleifen aus anderweitig deklarierten DATA–Zeilen gefüllt werden.

Beispiel 2:

```
DIM SHARED A,B,C(10,2)
DIM CF(19)
FOR I=1 TO 19
    READ CF(I)
    PRINT CF(I)
NEXT I
DATA 0,2,4,5,7,9,11,0,1,-1,0,0,0,0,0,0,0,-12,12,0
```

Anmerkungen:

- Mit DIM SHARED können auch einfache Variable als globale Variable deklariert werden (s. Beispiel 2). Das erspart ggf. separate SHARED–Anweisungen in den jeweiligen Unterprogrammen.

- Der kleinste Indexwert für ein Feld ist immer Null, es sei denn, mit der OPTION BASE–Anweisung (s. dort) wurde etwas anderes vereinbart. DIM A(10) reserviert also Platz für 11 Elemente.

- Der größte Indexwert für ein Feld ist 32767.

- Es sind maximal 255 Dimensionen möglich; es ist jedoch die Hauptspeicherkapazität zu berücksichtigen.

- Ohne DIM–Anweisung vereinbart der BASIC–Interpreter bei der ersten Verwendung einer Feldvariablen automatisch einen größten Indexwert von 10, erlaubt also Felder von maximal 11 Elementen ohne Dimensionierung.

- Verwendung von größeren als dimensionierten Indizes führt zu der Fehlermeldung

  Subscript out of range

  (Index außerhalb des Bereiches).

- Mehrmaliges Dimensionieren derselben Feldvariablen führt zur Fehlermeldung

  Duplicate definition

  (Doppelte Definition).

END–Anweisung

Format: **END**

Bedeutung: Alle ggf. geöffneten Dateien werden geschlossen, das Programm wird beendet, und Amiga Basic kehrt auf die Befehlsebene (Direkt–Modus) zurück.

Beispiel:

```
Warte:
Z$=INKEY$
IF Z$="*" THEN END ELSE Warte
```

Das Programm wartet auf das Drücken einer Taste und wird nur beendet, wenn die *–Taste gedrückt wird.

Anmerkungen:

- Es können beliebig viele END–Anweisungen an beliebigen Stellen im Programm stehen.
- In einem linearen Programm braucht als letzte auszuführende Anweisung keine END–Anweisung zu stehen.
- Programme, die mit END beendet wurden, können mit der CONT–Anweisung (s. dort) fortgesetzt werden, falls nach der END–Anweisung noch weitere Anweisungen folgen.

END SUB–Anweisung

Siehe SUB–Anweisung.

EOF–Funktion

Format: ***v*= EOF(*Dateinr*)**

Bedeutung: Die EOF–Funktion liefert einen logisch "wahren" Wert (–1), wenn bei einer Leseoperation in einer sequentiellen Datei das Dateiende erreicht wurde und einen logisch "falschen" Wert (0), wenn noch kein Ende erreicht wurde.

Dateinr Die Nummer, unter der die Datei mit OPEN eröffnet wurde.

Beispiel:

```
OPEN "I",#3,"EINGABE"
WHILE NOT EOF(3)
    LINE INPUT #3,A(J)
    J=J+1
WEND
CLOSE #3
END
```

Es werden solange numerische Daten aus der sequentiellen Eingabedatei EINGABE gelesen und im Feld A abgelegt, bis das Dateiende erreicht wird.

Anmerkungen:

- Mit der EOF–Funktion kann die Fehlermeldung

  ```
  Input past end
  ```

  (Eingabe nach logischem Dateiende) vermieden werden.

ERASE–Anweisung

Format: **ERASE** ***Feldvar***[,***Feldvar***]

Bedeutung: Es werden die Felder, deren Namen als Parameter in der Anweisung angegeben werden, aus dem Programm gelöscht und der dafür reservierte Speicherplatz wird wieder freigegeben.

Feldvar Der Name des zu löschenden Feldes.

Beispiel:

```
CLEAR ,100000
START=FRE(0)
DIM FELD(100,100)
MITTE=FRE(0)
ERASE FELD
DIM FELD (10,10)
ENDE=FRE(0)
PRINT START;MITTE;ENDE
```

Wird dieses Programm gestartet, so erhält man im Ausgabefenster eine Anzeige wie:

```
99984 59145 99450
Ok
```

Nach dem Löschen des sehr groß dimensionierten Feldes FELD in der 5. Zeile wird dieses neu dimensioniert. Jeweils vor und nach den Dimensionierungen wird die Anzahl der verbleibenden Speicherplätze vermerkt und am Ende ausgedruckt.

Anmerkungen:

- Nachdem ein Feld mit ERASE gelöscht wurde, kann es mit DIM neu dimensioniert werden.

ERL–Systemvariable

Format: ***v*=ERL**

Bedeutung: Es wird die Nummer der Programmzeile übergeben, in der ein Fehler aufgetreten ist. Hat die fehlerhafte Zeile keine Zeilennummer, so liefert ERL die Nummer der letzten vor der fehlerhaften Zeile numerierten Zeile. ERL liefert **nicht** den Namen einer alphanumerischen Sprungmarke.

Beispiel:
```
200 IF ERL=100 THEN 1000
```

Anmerkungen:
- Wenn eine fehlerhafte Anweisung im Direktmodus ausgeführt werden soll, weist der Interpreter der ERL–Variablen den Wert 65535 zu.

ERR–Systemvariable

Format: ***v*=ERR**

Bedeutung: Der Code des jeweils letzten vom Interpreter diagnostizierten Fehlers (s.a. Anhang B) wird der Systemvariablen ERR zugewiesen (s.a. ON ERROR–Anweisung).

Beispiel:

```
ON ERROR GOTO Fehler
    .
    .
    .
Fehler: IF ERR<>11 THEN . . .
PRINT "Division durch Null": END
```

Im Fehlerfall verzweigt das Programm zur Sprungmarke Fehler, wo geprüft wird, ob der Fehler durch eine Division durch Null verursacht wurde. Falls ja, wird eine Meldung angezeigt und das Programm beendet.

ERROR–Anweisung

Format: **ERROR *n***

Bedeutung: Es wird entweder ein Programmfehler simuliert oder es wird ein anwender–spezifischer Fehlercode definiert.

n Wird für ***n*** (0 bis 255) ein gültiger Fehlercode (s. Anhang B) angegeben, so simuliert ERROR den dadurch definierten Fehler. Existiert eine Verzweigung in eine Fehlerroutine (s. ON ERROR–Anweisung), so wird die Fehlerroutine durchlaufen. Andernfalls wird die Fehlermeldung angezeigt, und das Programm wird angehalten. Für die Definition eigener Fehlercodes müssen für ***n*** Werte außerhalb der Fehlerliste von Amiga Basic (s. Anhang B) angegeben werden (z.B. über 150). So definierte Fehler werden genau wie Standard–Fehlermeldungen verarbeitet. Tritt ein selbstdefinierter Fehler auf, und es existiert dafür keine Fehlerroutine, so meldet der Interpreter

```
Unprintable error
```

(nicht druckbarer Fehler) und hält das Programm an.

Beispiel:

```
E=23:ERROR E
```

Wird diese Zeile ausgeführt, so wird ein Kommunikationsfenster mit der Meldung

```
Line Buffer Overflow
```

angezeigt.

Beispiel 2:

```
100 ON ERROR GOTO 1000
110 INPUT "Bitte Anfangswert angeben";AW
120 IF AW<12 THEN ERROR 150
    .
    .
    .
1000 IF ERR=150 THEN PRINT "zu klein"
1010 IF ERL=120 THEN RESUME 110
```

Wird in Zeile 110 für AW ein kleinerer Wert als 12 eingegeben, so führt dies zu einem selbstdefinierten Fehler (Code 150), der in Zeile 1000 bearbeitet wird und zur Eingabewiederholung führt.

EXIT SUB–Anweisung

Siehe SUB–Anweisung.

EXP–Funktion

Format: ***v*= EXP(*x*)**

Bedeutung: Die EXP–Funktion berechnet den Wert der Exponentialfunktion zur Basis e mit dem numerischen Ausdruck ***x*** im Exponenten.

Beispiel:

```
Ok
X=4:PRINT EXP(SQR(X))
 7.389056
Ok
```

Es wird e hoch Wurzel aus 4, also e zum Quadrat, berechnet.

Anmerkungen:

- Falls ein numerischer Wert größer als 88 bei einfacher Genauigkeit sowie 709 bei doppelter Genauigkeit angegeben wird, zeigt der Interpreter den Fehler

  ```
  Overflow
  ```

  (Überlauf) an.

FIELD–Anweisung

Format: **FIELD** [#]***Dateinr,Länge* AS *Zeikettvar***
[,***Länge* AS *Zeikettvar***]. . .

Bedeutung: Es wird Platz für Variablenwerte in einem Puffer für Dateien mit wahlfreiem Zugriff reserviert.

Dateinr Die Nummer, unter der die Datei für wahlfreien Zugriff eröffnet wurde.

Länge Ein numerischer Ausdruck, dessen Wert die für die jeweils mit angegebene Zeichenkettenvariable zu reservierende Länge (Zeichenanzahl) angibt.

Zeikettvar Name einer Zeichenkettenvariablen, der die definierte Zahl von Zeichen aus dem Puffer zugewiesen wird (s.a. GET– und PUT–Anweisungen für Dateien).

Beispiel:

```
OPEN "LAGER" AS #3 LEN=144
FIELD #3,2 AS ARTNR$,20 AS BEZ$,2 AS STZ$
FOR I=1 TO 100
    LSET ARTNR$=MKI$(ANZ(I))
    LSET BEZ$=BZ$(I)
    LSET STZ$=SZ$(I)
    PUT #3,I
NEXT
```

Bei diesem Beispiel wird zunächst die Datei LAGER für wahlfreien Zugriff auf dem aktuellen Laufwerk eröffnet. Dann wird jeder logischen Datensatz in 3 Felder unterschiedlicher Länge unterteilt, die auch benannt werden. In einer Schleife werden dann 100 Sätze in die Datei geschrieben, wobei für jeden Satz die Daten aus anderweitig dimensionierten und gefüllten Feldern entnommen und vor dem Schreiben in dem Puffer abgelegt werden (s.a. Kapitel 5).

Anmerkungen:

- Die FIELD–Anweisung überträgt **keine** Daten in den Puffer (s. dazu LSET– und RSET–Anweisungen).

- Die FIELD–Anweisung liest keine Daten aus der Datei, sondern unterteilt den Puffer in Felder und ordnet diese den angegebenen Variablen zu.

- Die Summe der für eine Datei in einer FIELD–Anweisung definierten Feldlängen darf die durch die OPEN–Anweisung festgelegte logische Datensatzlänge nicht überschreiten. Andernfalls wird die Meldung

  ```
  Field overflow
  ```

  (Feldüberlauf) angezeigt.

- Unter einer Dateinummer dürfen beliebig viele FIELD–Anweisungen deklariert werden, die alle gültig sind. Jede neue FIELD–Anweisung beginnt aber immer mit der ersten Position im Puffer.

- **Achtung:** Wird eine in einer FIELD–Anweisung definierte Variable später im Programm bei einer Zuweisung (z.B. A$ = "DATEN") oder in einer INPUT–Anweisung (z.B. INPUT "Bitte Eingabe";A$) verwendet, so ist damit die Zuweisung in der Fieldanweisung aufgehoben.

FILES–Befehl

Format: **FILES** [***Dateiangabe***]

Bedeutung: Es wird der Inhalt des aktuellen Diskettenverzeichnisses auf dem Bildschirm angezeigt. Der Befehl entspricht dem DIR–Befehl im AmigaDOS.

Dateiangabe Ein Zeichenkettenausdruck wie in Kapitel 5.2 angegeben. Wird die Dateiangabe im FILES–Befehl weggelassen, so werden alle Dateien des aktuellen Diskettenverzeichnisses angezeigt.

Beispiel 1:

```
FILES
```

Alle Dateien des aktuellen Diskettenverzeichnisses auf dem voreingestellten Laufwerk werden angezeigt.

Beispiel 2:

```
FILES "DF1:"
```

Alle Dateien im Basis-Verzeichnis auf Laufwerk DF1: werden angezeigt.

Anmerkungen:

- Im Kapitel 1.3.3 des AmigaDOS–Benutzerhandbuches sind die Datei– und Verzeichnis–Bennungskonventionen ausführlich beschrieben.

FIX–Funktion

Format: ***v*= FIX(*x*)**

Bedeutung: Die Dezimalstellen des Wertes des beliebigen numerischen Ausdrucks ***x*** rechts vom Dezimalpunkt werden ohne Rundung abgeschnitten.

x Ein beliebiger numerischer Ausdruck.

Beispiele:

```
Ok
PRINT FIX(3.84)
 3
Ok
PRINT FIX(-12.85)
-12
Ok
PRINT INT(-12.85)
 13
Ok
```

Anmerkungen:

- Bei negativen Werten wird **nicht** die nächst kleinere Zahl übergeben (s.a. INT– und CINT–Funktionen).

FOR– . . . NEXT–Anweisungen

Format: **FOR** *Var*= *x* **TO** *y* [**STEP** *z*]

.
.
.

NEXT [*Var*][,*Var*]

Bedeutung: Die Kombination dieser beiden Anweisungen erlaubt die Mehrfachverarbeitung einer Folge von Befehlen, Anweisungen und/oder Funktionen in einer Schleife mit einer definierten Zahl von Durchläufen.

Var Eine beliebige ganzzahlige oder einfach–genaue Variable, die als Schleifendurchlaufzähler dient.

x Ein numerischer Ausdruck, dessen Wert den Zähleranfangswert festlegt.

y Ein numerischer Ausdruck, dessen Wert den Zählerendwert festlegt.

z Ein numerischer Ausdruck, dessen Wert die Schrittweite festlegt, um die der Durchlaufzähler nach jedem Durchlauf erhöht wird. Wird ***z*** nicht angegeben, so wird eine Schrittweite von 1 angenommen.

Beispiel 1:

```
J=12:K=50:L=4
FOR I=1 TO J STEP L
    PRINT I;K+1
NEXT
```

ergibt in dem Ausgabefenster

```
1 51
5 55
9 59
Ok
```

Die Schleife mit einer Schrittweite von 4 wird nur dreimal durchlaufen, weil der Schleifenzähler I beim 4. Durchlauf größer als der Endwert J würde.

Beispiel 2:

```
FOR I=1 TO 10
   FOR J=1 TO 3
      READ B(I,J)
NEXT J,I
```

Bei diesem Beispiel wird in einer geschachtelten Schleife das Feld B aus anderweitig definierten DATA–Anweisungen gefüllt.

Anmerkungen:

- Vor der Ausführung der NEXT–Anweisung wird nach der Erhöhung des Schleifenzählers um die Schrittweite geprüft, ob dieser größer als der vorgegebene Endwert ist. Ist er größer, fährt das Programm mit der auf NEXT folgenden Anweisung fort; ist er kleiner oder gleich, wird die Schleife erneut durchlaufen.

- Bei negativer Schrittweite muß der Anfangswert größer als der Endwert angegeben werden. Die Schleife wird dann solange durchlaufen bis der Zähler kleiner als der Anfangswert wird.

- Ist die Schrittweite 0, so wird die Schleife unendlich oft durchlaufen, es sei denn, sie wird durch eine Programmabfrage (z.B. IF. . .THEN) verlassen.

- Bei geschachtelten Schleifen muß für jeden Zähler eine eigene Variable angegeben werden. Die NEXT–Anweisung für eine innere Schleife muß immer **vor** der NEXT–Anweisung der sie umgebenden äußeren Schleife im Programm stehen. Die Schreibweise NEXT *Var1,Var2,Var3*. . . ist ebenso zulässig wie die Variante NEXT *Var1*:NEXT *Var2*:NEXT *Var3*. . .

- Variablenangaben in der NEXT–Anweisung können weggelassen werden. Sie erhöhen jedoch speziell bei mehrfach geschachtelten Schleifen die Lesbarkeit des Programms.

- Fehlen bei einer Schleife die FOR– oder die NEXT–Anweisung, so werden die Fehlermeldungen

  ```
  FOR without NEXT
  ```

  (FOR ohne NEXT) bzw.

  ```
  NEXT without FOR
  ```

  (NEXT ohne FOR) ausgegeben.

FRE–Funktion

Format: ***v*=FRE(*x*)**

Bedeutung: Die FRE–Funktion liefert je nach Wert des numerischen Argumentes ***x*** verschiedene Aussagen über freie Speicherplätze in Bytes:

$x=-1$	noch verfügbarer Systemspeicher;
$x=-2$	noch verfügbarer Amiga Basic–Stapelspeicher;
$x>-1$	noch verfügbarer Speicher im Amiga Basic–Datensegment.

Beispiel:

```
PRINT FRE(1);FRE(-1)
 20350  225400
Ok
```

Der verfügbare Speicherplatz im Amiga Basic–Datensegment, also für Basic–Programmtext, beträgt 20350 Bytes, während im System noch 225400 Bytes unbenutzt sind.

Anmerkungen:

- Ehe mit der CLEAR–Anweisung (s. dort) mehr Basic–Speicher zugewiesen wird, sollte mit der FRE–Funktion der noch verfügbare Systemspeicher überprüft werden.

GET–Anweisung für Dateien

Format: **GET** [#]***Dateinr***[,***Satznr***]

Bedeutung: Der logisch nächste oder der angegebene Satz wird aus einer Datei für wahlfreien Zugriff (Direktzugriffsdatei) in einen Puffer übernommen.

Dateinr Die Nummer, unter der die Datei in der OPEN–Anweisung eröffnet wurde.

Satznr Die Nummer des zu lesenden logischen Datensatzes. Die Nummer muß zwischen 1 und 16777215 liegen, wenn sie angegeben wird. Wird sie nicht angegeben, so wird der logisch nächste Satz der Datei gelesen.

Beispiel 1:

```
OPEN "LAGER" AS #3 LEN=144
FIELD #3,2 AS ARTNR$,20 AS BEZ$,2 AS STZ$
GET #3
PRINT ARTNR$,STZ$
    .
    .
    .
```

Nach der Eröffnung der Datei LAGER für wahlfreien Zugriff werden die einzelnen Felder der zu lesenden Sätze definiert und benannt. Anschließend wird der logisch nächste Satz aus der Datei in den Puffer gelesen, und 2 Elemente dieses Satzes werden ausgedruckt.

Anmerkungen:

- Nachdem GET ausgeführt wurde, können mit INPUT #, LINE INPUT # oder einfach durch Bezugnahme auf in der FIELD-Anweisung deklarierten Variablen Daten aus dem Dateipuffer übernommen werden (s.a. Kapitel 5.3).

- Amiga-DOS legt auf Diskette oder Festplatte physikalische Sätze (Sektoren) von 512 Bytes Länge an, in die fortlaufend die logischen Sätze übertragen werden, so daß bei kürzeren Satzlängen mehrere logische Sätze in einen physikalischen Satz passen. Deshalb führt eine GET–Anweisung nicht unbedingt zu einem physikalischen Lesen von Disk.

GET–Anweisung für Grafik

Format: **GET (*x1,y1*)–(*x2,y2*),*Feldvar* [(*Index*[,*Index*. . .,*Index*])]**

Bedeutung: Es werden von einem definierten Bildschirmbereich (Fenster) die Bildpunkte gelesen und als binäre Information (bitweise) in einem numerischen Feld gespeichert.

(x1,y1),(x2,y2) Absolute Koordinaten der oberen linken und unteren rechten Ecke des auszulesenden Bereiches.

Feldvar Eine numerische Feldvariable beliebiger Genauigkeit, in der die Bildpunktinformation bitweise abgelegt wird.

Index Die optionale Angabe von Feldelementen über deren Index erlaubt die Speicherung mehrere grafischer Objekte in einem mehrdimensionalen Feld. Auf diese Weise können verschiedene Ansichten des Objektes gespeichert und auch schnell mit der PUT–Anweisung für Grafik (s. dort) wieder dargestellt werden.

Die Anzahl der erforderlichen Bytes in dem Feld errechnet sich aus

$6+(y2-y1+1)*2*\mathrm{INT}((x2-x1+16)/16)*T$

wobei *x1,y1* bzw. *x2,y2* die Koordinaten der linken oberen bzw. rechten unteren Ecke des Bildausschnittes bedeuten. *T* ist die Bildschirmtiefe, die mit 2 voreingestellt ist (s. SCREEN–Anweisung).

Beispiel:

```
DIM A%(26)
GET (0,0)-(9,11),A%
```

In diesem Beispiel soll ein Bildfenster von 10x12 Punkten in das Ganzzahlfeld A% gespeichert werden. Nach der oben beschriebenen Formel werden dazu

$6+(11-0+1)*2*\mathrm{INT}((9-0+16)/16)*2 = 54$ Bytes

benötigt. In einem Ganzzahlfeld werden 2 Bytes pro Element reserviert, so daß ein Feld von 27 Elementen (0 bis 26) ausreicht.

Anmerkungen:

- Wenn ein Ganzzahlfeld für die Bildinformation angelegt wird, so enthält das Feldelement 0 die Breite, Feldelement 1 die Höhe, Feldelement 2 die Tiefe und die anschließenden Elemente die binären grafischen Informationen des rechteckigen Bildausschnittes. In diesem Fall können also die grafischen Daten direkt im Feld interpretiert werden.

- So gespeicherte Informationen können mit der PUT–Anweisung für Grafik (s. dort) schnell wieder auf den Bildschirm gebracht werden.

GOSUB– und RETURN–Anweisungen

Format:

GOSUB ***Marke***

.
.
.

RETURN [***Weiter***]

Sprung in eine Subroutine und Rückkehr in das Hauptprogramm (s.a. ON. . .GOSUB–Anweisung).

Marke,Weiter Eine gültige Zeilennummer oder alphanumerische Sprungmarke, bei der die Subroutine beginnt bzw. bei der nach der Rückkehr aus der Subroutine das Hauptprogramm fortgeführt werden soll. Wird bei RETURN ***Weiter*** nicht angegeben, so wird das Hauptprogramm mit der auf die GOSUB–Anweisung folgenden Anweisung fortgesetzt.

Beispiel:

```
100 PRINT "Dies ist das Hauptprogramm"
110 GOSUB 140
120 PRINT "Fortsetzung d. Hauptprogramms"
130 END
140 PRINT "Vom Unterprogramm gedruckt"
150 RETURN
```

Dieses Programm ergibt im Ausgabfenster folgende Anzeige:

```
Dies ist das Hauptprogramm
Vom Unterprogramm gedruckt
Fortsetzung d. Hauptprogramms
Ok
```

Anmerkungen:

- Bei Amiga Basic wird zwischen Subroutinen und Unterprogrammen unterschieden. Subroutinen sind immer in Amiga Basic programmiert, werden ausschließlich mit GOSUB aufgerufen und kennen keine lokalen Variablen. Unterprogramme können in Amiga Basic oder auch in Maschinensprache programmiert sein, kennen lokale und globale Variablen und werden mit der CALL–Anweisung (s. dort und Kapitel 6.1) aufgerufen.

- Es dürfen beliebig viele Aufrufe für eine Subroutine in einem Programm vorkommen.

- Subroutinen können an beliebiger Stelle im Hauptprogramm stehen, sollten jedoch zusammengefaßt werden und vom Hauptprogramm durch eine GOTO– oder END–Anweisung getrennt werden, damit sie nicht unkontrolliert durchlaufen werden.

- Die Schachtelungstiefe (Aufruf einer Subroutine aus einer Subroutine) wird nur durch die Hauptspeicherkapazität begrenzt.

- Die Anweisung RETURN übergibt die Programmsteuerung an die Anweisung, die im Hauptprogramm auf die GOSUB-Anweisung folgt, oder an die Zeile im Hauptprogramm, deren Nummer oder alphanumerische Sprungmarke bei der RETURN–Anweisung angegeben wurde.

- Bei der Angabe einer Zeilennummer oder alphanumerischen Sprungmarke bei der RETURN–Anweisung ist mit Vorsicht vorzugehen, da alle zur Zeit des Subroutinenaufrufes aktiven anderen GOSUB–, CALL–, WHILE– und FOR–Anweisungen auch hinterher aktiv bleiben, also Bezugsadressen auf dem Stapelspeicher hinterlegt haben.

- Es dürfen beliebig viele RETURN–Anweisungen in einer Subroutine stehen.

GOTO–Anweisung

Format: **GOTO *Marke***

Bedeutung: Unbedingte Programmverzweigung zu einer bestimmten Programmzeile (s.a. ON. . .GOTO–Anweisung).

Marke Eine gültige Zeilennummer oder alphanumerische Sprungmarke, die die Zeile bezeichnet, bei der das Programm fortgesetzt werden soll.

Beispiel:

```
CheckMaus:
    IF MOUSE(0)=0 THEN CheckMaus
    IF ABS(X-MOUSE(1))>2 THEN SchiebBild
    IF ABS(Y-MOUSE(2))<3 THEN CheckMaus
SchiebBild:
    PUT (X,Y),P
    X=MOUSE(1):Y=MOUSE(2)
    PUT (X,Y),P
    GOTO CheckMaus
```

Ein Beispiel, wie mit Hilfe der GOTO–Anweisung ein kleines Programm erstellt werden kann, mit dem ein grafisches Objekt mit der Maus bei gedrückter Auswahltaste auf dem Bildschirm bewegt werden kann.

Anmerkungen:

- Enthält die hinter GOTO angegebene Zeile nicht ausführbare Anweisungen (DATA, REM), so wird das Programm mit der nächsten darauf folgenden, ausführbaren Zeile fortgesetzt.

- Im Direktmodus kann mit GOTO in ein Programm, das im Hauptspeicher resident ist, hineinverzweigt werden (z.B. zum Testen).

- Es empfiehlt sich, Programmsteuerstrukturen besser mit den Anweisungen IF. . .THEN. . .ELSE, WHILE. . .WEND oder ON. . .GOTO (s. dort) anstatt mit GOTO aufzubauen, da im letzteren Fall die Programme sehr unübersichtlich werden und schlecht zu testen sind.

HEX$–Funktion

Format: ***v$*** **= HEX$(*n*)**

Bedeutung: Es wird eine Zeichenkette übergeben, die die hexadezimale Darstellung des Wertes des numerischen Ausdrucks ***n*** enthält.

n Beliebiger numerischer Ausdruck, dessen ganzzahliger Wert gewandelt wird und der im Bereich zwischen –32768 und 65535 liegen muß. Bei negativen Werten wird die Zweier–Komplement–Darstellung benutzt.

Beispiel:

```
X=23.4:Y=-256
PRINT X;HEX$(X),Y;HEX$(Y)
```

ergibt im Ausgabefenster die Anzeige

```
23.4 17                -256 FF00
```

Anmerkungen:

- Vor der Wandlung werden ggf. vorhandene Dezimalstellen rechts vom Dezimalpunkt abgeschnitten.
- Die Wandlung in oktale Darstellung kann mit der OCT$-Funktion (s. dort) erreicht werden.

IF–Anweisung

Format:

Syntax 1: **IF *Ausdr* GOTO *Marke* [ELSE *Anweis*]**

Syntax 2: **IF *Ausdr* THEN *Anweis* [ELSE *Anweis*]**

Syntax 3:
IF *Ausdr* THEN *Anweis*
ELSEIF *Ausdr* THEN *Anweis*
ELSE *Anweis*
END IF

Bedeutung: Erlaubt die Verzweigung in verschiedene Programmteile oder die Ausführung verschiedener Anweisungsblöcke, abhängig vom Wahrheitsgehalt eines logisch auswertbaren Ausdruckes. Ein solcher Ausdruck ist logisch "wahr" (s.a. Kapitel 8.6), wenn sein Wert von Null verschieden ist, und logisch falsch, wenn sein Wert Null ist.

Ausdr Jeder beliebige Ausdruck, der logisch auswertbar ist.

Anweis Eine einzelne Anweisung oder ein aus mehreren durch Doppelpunkte getrennten Anweisungen zusammengesetzter Anweisungsblock oder auch eine gültige Amiga Basic–Zeilennummer oder alphanumerische Sprungmarke, die die Zeile bezeichnet, zu der nach THEN oder ELSE verzweigt werden soll.

Marke Eine gültige Amiga Basic–Zeilennummer oder alphanumerische Sprungmarke, die die Zeile bezeichnet, zu der verzweigt werden soll.

Beispiel:

```
INPUT a,b
IF a=1 THEN
    IF b=1 THEN
        PRINT "a=1 und b=1"
    ELSE
        PRINT "a=1, b<>1"
    END IF
ELSEIF a>0 THEN
    IF b>0 THEN PRINT "a>0 und b>0"
    REM *** obige Zeile ist keine Block-IF-Anweisung
    PRINT "a>0"
ELSE
    PRINT "a<=0"
    PRINT "über b ist nichts bekannt"
END IF
```

Anmerkungen: Folgende Regeln gelten für alle 3 Formen der IF–Anweisung:

- Ist das Ergebnis von ***Ausdr*** logisch "wahr", wird GOTO oder ***Anweis*** hinter THEN ausgeführt.

- Ist das Ergebnis von ***Ausdr*** logisch falsch, wird GOTO oder ***Anweis*** hinter THEN ignoriert und stattdessen ***Anweis*** hinter ELSE, falls vorhanden, ausgeführt.

- ***Anweis*** hinter THEN oder ELSE kann aus mehreren Amiga Basic–Anweisungen oder –Funktionen bestehen. Diese dürfen bei Syntax 1 und 2 zusammen mit der IF–Anweisung jedoch die Länge einer Amiga Basic–Programmzeile nicht überschreiten.

- Hinter THEN können entweder Amiga Basic–Anweisungen oder –Funktionen oder aber auch eine Zeilennummer oder alphanumerische Sprungmarke folgen.

- Hinter GOTO muß auf jeden Fall eine Zeilennummer oder alphanumerische Sprungmarke folgen.

- Wenn die gesamte IF–Anweisung nicht dieselbe Anzahl von THEN– und ELSE–Klauseln enthält, bezieht sich jede ELSE-Klausel immer auf die **nächste** unerfüllte THEN–Klausel.

- Wenn im Direktmodus bei einer IF. . .THEN–Anweisung eine Zeilennummer oder alphanumerische Sprungmarke angegeben wird, wird die Fehlermeldung

  Undefined label

  (undefinierte Marke) bei logisch wahrem Ausdruck angezeigt, es sei denn, daß vorher im Edier–Modus eine solche Nummer bzw. Sprungmarke eingegeben wurde.

- Ein IF–Block braucht nicht die erste Anweisung in einer Programmzeile zu sein.

Die Anweisungssyntax 3 unterscheidet sich von den anderen beiden in folgenden Punkten:

- Hier kann ***Anweis*** geschachtelte IF. . .THEN. . .ELSE–Blöcke enthalten. Geschachtelte Anweisungen sind bei Amiga Basic **nicht** auf eine logische Programmzeile beschränkt. ***Anweis*** kann hier also aus mehreren Anweisungen in aufeinanderfolgenden logischen Programmzeilen bestehen.

- Ist ***Ausdr*** logisch "wahr", wird ***Anweis*** hinter THEN ausgeführt und das Programm dann mit der ersten auf die END IF–Anweisung folgenden Anweisung fortgesetzt.

- Ist keiner der numerischen Ausdrücke logisch "wahr", so wird:

  entweder das Programm mit der ersten auf die END IF–Anweisung folgenden Anweisung fortgesetzt,

  oder ***Anweis*** hinter ELSE (falls vorhanden) ausgeführt und das Programm mit der ersten auf die END IF–Anweisung folgenden Anweisung fortgesetzt.

- Der ELSEIF–Block ist optional. Es können beliebig viele ELSEIF–Blöcke angegeben werden.

- Der ELSE–Block ist ebenfalls optional.

- Folgt hinter THEN eine andere Anweisung als REM oder ein durch ' eingeleiteter Kommentar auf derselben Programmzeile, so wird dies von Amiga Basic als eine einzelne IF. . . THEN. . .ELSE–Anweisung interpretiert.

- In einer Programmzeile mit einem ELSE–, ELSEIF– oder END IF–Block darf der Anweisung nur eine alphanumerische Sprungmarke vorangestellt werden. Andernfalls wird eine Fehlermeldung erzeugt.

INKEY$–Funktion

Format: ***v$*** **= INKEY$**

Bedeutung: Es wird ein Zeichen aus dem Tastaturpuffer übergeben. Ist der Tastaturpuffer leer, so wird eine Leer–Zeichenkette (Länge Null) übergeben.

Beispiel 1:

```
Taste:
a$=INKEY$
IF a$<>"" THEN a$=UCASE$(A$)
    IF A$="J" THEN Antwort=1
    IF A$="N" THEN Antwort=2
    IF A$="C" THEN Antwort=3
    IF Antwort=0 THEN BEEP
END IF
IF Antwort=0 THEN GOTO Taste
PRINT Antwort
```

Das Programm wartet auf das Drücken einer Taste. Wurde eine gedrückt, so wird, falls Buchstabe, dieser in Großbuchstabe gewandelt. Ist es J, N oder C, wird eine Antwort ausgedruckt, sonst ein Piepton erzeugt und wieder gewartet.

Anmerkungen:

- INKEY$ kann Zeichen grundsätzlich nur an Zeichenkettenvariablen übergeben.
- INKEY$ zeigt grundsätzlich kein Zeichen an, sondern übergibt es nur an das Programm.
- Alle Zeichen werden von INKEY$ an das Programm weitergereicht, mit Ausnahme von Amiga–. oder CTRL–C; diese Kombinationen unterbrechen das Programm.
- Wenn bei ablaufendem Programm kein Ausgabefenster aktiv ist, und bei der INKEY$–Anweisung eine Taste gedrückt wird, wird der zugeordnete Code ignoriert und ein Piepton erzeugt, da beim Amiga Tastatureingaben nur an ein aktiviertes Ausgabefenster weitergegeben werden.
- Eine weitere Möglichkeit, Programme bis zum Eintritt eines bestimmten Ereignisses anzuhalten, ist die SLEEP–Anweisung (s. dort).

INPUT–Anweisung

Format: **INPUT**[;] ["*Text*";]*Var*[,*Var*]. . .

Bedeutung: Es werden Tastatureingaben während der Programmausführung an Variablen übergeben.

Text Eine beliebige Zeichenkettenkonstante, die vor der Tastatureingabe angezeigt wird und diese z.B. näher erläutert.

Var Beliebige numerische oder Zeichenkettenvariable oder ein Feldelement, die oder das den Wert der Tastatureingabe (numerisch oder Zeichenkette) zugewiesen bekommt.

Die INPUT–Anweisung gibt bei der aktuellen Cursor–Position den ggf. in der Anweisung definierten Text sowie immer ein Fragezeichen (?) aus und erwartet dann die Tastatureingabe (Cursor erscheint im aktuellen Ausgabefenster), die mit der Return–Taste abgeschlossen wird.

Beispiel 1:

```
INPUT I%,J%
PRINT I%,J%
```

Wird dieses kleine Programm gestartet, so müssen im Ausgabefenster nach dem Fragezeichen zwei Zahlen eingegeben werden, die dann wieder angezeigt werden, also z.B.:

```
? 123,456
 123     456
Ok
```

Beispiel 2:

```
PI=3.14159
INPUT "Durchmesser eingeben";D
   U=PI*D
   PRINT "Der Kreisumfang ist";U
END
```

Hier wird der Eingabeaufforderung ein Text vorangestellt, also:

```
Durchmesser eingeben ? 20
Der Kreisumfang ist 62.8318
```

Anmerkungen:

- Die Anzeige des Fragezeichens wird unterdrückt, wenn hinter *Text* anstelle des Semikolons (;) ein Komma (,) gesetzt wird.

- Werden mehrere Variablen in der INPUT–Anweisung deklariert, so müssen die einzelnen Eingaben durch Komma (,) getrennt werden.

- Die eingegebenen Daten müssen dem Typ der Variablen entsprechen, der sie zugeordnet werden sollen. Zeichenketten müssen dann in Anführungszeichen (") eingeschlossen werden, wenn sie signifikante führende oder nachfolgende Leerstellen oder Kommata enthalten.

- Stimmen Typ oder Anzahl der eingegebenen Datenelemente nicht mit den deklarierten Variablen überein, wird mit der Meldung

  ```
  ?Redo from start
  ```

  eine Wiederholung der gesamten Eingabe gefordert. Die Zuordnung zu den Variablen erfolgt erst nach der Akzeptierung aller Eingaben.

- Nach dem Drücken der Return–Taste ist die aktuelle Cursor–Position die Stelle nach dem zuletzt eingegebenen Zeichen, wenn unmittelbar nach dem Schlüsselwort INPUT ein Semikolon (;) gesetzt wird.

INPUT$–Funktion

Format: ***v$*** = **INPUT$(*n*[,[#]*Dateinr*])**

Bedeutung: Es wird eine ***n*** Zeichen lange Zeichenkette von der Tastatur oder einer sequentiellen Datei oder Einheit übernommen und einer Zeichenkettenvariablen zugewiesen.

n Ein beliebiger numerischer Ausdruck, dessen ganzzahliger Wert zwischen 1 und 32767 liegen muß und der die Zahl der zu lesenden Zeichen angibt.

Dateinr Die Nummer, unter der die Datei zum sequentiellen Lesen eröffnet wurde. Wird die Dateinummer weggelassen, so wird von der Tastatur gelesen.

Beispiel:

```
OPEN "I",#3,"TEXT"
Lesen:
   IF EOF(3) THEN Fertig
   PRINT ASC(INPUT$(1,#3));
   GOTO Lesen
Fertig:
CLOSE:END
```

Von allen Zeichen in der Datei TEXT im aktuellen Diskettenlaufwerk wird der ASCII–Code auf dem Bildschirm angezeigt.

Anmerkungen:

- INPUT$ übernimmt nur die angegebene Anzahl von Zeichen. Beenden der Eingabe durch die Return–Taste ist nicht möglich.
- INPUT$ übernimmt alle Tastencodes mit Ausnahme von Amiga–.; mit dieser Tastenkombination kann INPUT$ abgebrochen werden.
- INPUT$ kennt keine Trennzeichen.
- Alle von INPUT$ übernommenen Zeichencodes werden nicht auf dem Bildschirm abgebildet.
- INPUT$ eignet sich besonders zur Übernahme von Daten von der Datenfernübertragungsschnittstelle, da hier alle ASCII–Codes vorkommen können.

INPUT #–Anweisung

Format: **INPUT #*Dateinr*,*Var*[,*Var*]...**

Bedeutung: Es werden Daten von einer sequentiellen Datei oder Einheit gelesen und der (den) in der Anweisung deklarierten Variablen zugewiesen.

Dateinr Die Nummer, unter der die Eingabedatei eröffnet wurde.

Var Der Name für eine beliebige Variable (numerisch, Zeichenkette oder Feldelement), der die Daten zugewiesen werden sollen.

Beispiel:

```
OPEN "TEXT" FOR INPUT AS #3
Lesen:
   IF EOF(3) THEN Fertig
   INPUT #3,T$:PRINT T$
   GOTO Lesen
Fertig:
   CLOSE
END
```

Es werden Zeichenketten aus der sequentiellen Datei TEXT gelesen und angezeigt, bis das Dateiende erreicht ist.

Anmerkungen:

- INPUT # liest neben sequentiellen Diskettendateien auch von der Datenfernübertragungsschnittstelle oder der Tastatur, wenn diese als Eingabeeinheiten eröffnet wurden (s. Kapitel 5.1 und OPEN–Anweisung).
- Für Datenformate und –typen gelten dieselben Regeln wie bei der INPUT–Anweisung (s. dort).
- Zeichenketten, die in Anführungszeichen (") eingeschlossen sind, dürfen keine weiteren Anführungszeichen enthalten.
- Trennzeichen bei numerischen Werten sind Leerstelle, Komma, Wagenrücklauf– oder Zeilenvorschub–Code.

- Trennzeichen bei Zeichenketten sind Komma, Wagenrücklauf– oder Zeilenvorschubcode. Fehlt ein solches Trennzeichen in der Zeichenkette, wird die Datenübernahme nach 32767 Zeichen abgebrochen.

- Wird während der Datenübernahme eines Wertes das Dateiende erreicht, so werden der bezogenen Variablen keine Daten zugewiesen.

- INPUT # darf auch bei Dateien mit wahlfreiem Zugriff benutzt werden, wenn diese sequentiell gelesen werden sollen.

- Weitere Einzelheiten zur Dateneingabe von Dateien sind in Kapitel 5.4 ausführlich erläutert.

INSTR–Funktion

Format: ***n*= INSTR([*n*,]*x$*,*y$*)**

Bedeutung: Die Funktion sucht und liefert die Position des ersten Auftretens einer Teilzeichenkette in einer Zeichenkette.

n Ein numerischer Ausdruck, dessen ganzzahliger Wert zwischen 1 und 32767 liegen muß und der die Startposition in der zu durchsuchenden Zeichenkette angibt. Wird ***n*** nicht angegeben, so beginnt die Suche bei Position 1.

x$ Beliebiger Zeichenkettenausdruck, dessen Wert die zu durchsuchende Zeichenkette darstellt.

y$ Beliebiger Zeichenkettenausdruck, dessen Wert die gesuchte Teilzeichenkette darstellt.

Beispiel:

```
IF LEN(A$)<47 THEN Weiter
IF INSTR(45,A$,"ABC")>0 GOTO Ausgabe
   PRINT "ABC nicht in "A$" enthalten"
   GOTO Weiter
Ausgabe:
   PRINT A$
   .
   .

Weiter:
   .
   .
   .
```

Der Text ABC soll ab dem 45. Zeichen in der Zeichenkette A$ gesucht werden. Ist A$ kürzer als 47 Zeichen, ist die Suche natürlich zwecklos.

Anmerkungen:

- In den folgenden Fällen liefert INSTR den Wert Null:

  n ist größer als LEN(***x$***)
  x$ ist leer
  y$*** ist nicht in ***x$ enthalten

- Ist ***y$*** leer, übergibt INSTR den Wert von ***n***, falls ***n*** angegeben war, andernfalls eine 1.

- Ungültige ***n***–Werte werden mit

  ```
  Illegal function call
  ```

  (unerlaubter Funktionsaufruf) quittiert.

INT–Funktion

Format: **v= INT(x)**

Bedeutung: Es wird der größte ganzzahlige Wert übergeben, der kleiner oder gleich *x* ist.

x Beliebiger numerischer Ausdruck.

Beispiel:

```
Ok
PRINT INT(12.18),INT(-14.23)
 12              15
Ok
```

Bei positiven Werten werden die Stellen rechts vom Dezimalpunkt abgeschnitten; bei negativen Werten wird abgerundet.

Anmerkungen:

- Die Funktionen FIX und CINT (s.dort) bilden ebenfalls ganzzahlige Werte, die jedoch auf andere Art ermittelt werden.

KILL–Befehl

Format: **KILL** ***Dateiangabe***

Bedeutung: Es wird die angegebene Datei auf Diskette oder Festplatte gelöscht.

Dateiangabe Ein Zeichenkettenausdruck für eine Dateispezifikation, wie in Kapitel 5.2 beschrieben.

Beispiel 1:

```
KILL "ADRESSEN"
```

Die Datei ADRESSEN wird im aktuellen Verzeichnis der Diskette im aktuellen Laufwerk gelöscht.

Beispiel 2:

```
KILL "DF1:LAGER/TEILE.DAT"
```

Die Datei TEILE.DAT im Unterverzeichnis LAGER auf der Diskette in Laufwerk DF1: wird gelöscht.

Anmerkungen:

- Der Löschversuch bei geöffneten Dateien führt zu der Fehlermeldung

  ```
  File already open
  ```

  (Datei bereits geöffnet.

LBOUND– und UBOUND–Funktionen

Formate: **LBOUND (*Feldname* [,*Dimension*])**
UBOUND (*Feldname* [,*Dimension*])

Bedeutung: Liefert die untere (LBOUND) bzw. obere (UBOUND) Grenze der angegebenen Dimension einer Feldvariablen.

Feldname Name der zu untersuchenden Feldvariablen.

Dimension Ganzzahliger Ausdruck, der die Nummer der Dimension angibt, deren untere bzw. obere Begrenzung ermittelt werden soll. Voreingestellt ist hier 1. Dieser Parameter ist optional und für mehrdimensionale Felder vorgesehen.

Beispiel:

```
CALL INKREMENT (FELD1(0),FELD2(),SUM())
    .
    .
    .
SUB INKREMENT (A(2),B(2),C(2)) STATIC
    FOR I=LBOUND(A,1) TO UBOUND(A,1)
        FOR J=LBOUND(A,2) TO UBOUND(A,2)
        C(I,J)=A(I,J)+B(I,J)
        NEXT J
    NEXT I
END SUB
```

Anmerkungen:

- Die LBOUND– und UBOUND–Funktionen sind besonders nützlich zur Bestimmung der Größe eines Feldes, das an ein Unterprogramm übergeben werden soll.

- Die untere Begrenzung ist der kleinste Index für die angegebene Dimension des Feldes. LBOUND liefert hier entweder 0 oder 1, je nachdem, was mit der OPTION BASE–Anweisung (s. dort) vereinbart wurde.

LEFT$–Funktion

Format: ***v$*** **= LEFT$(*x$,n*)**

Bedeutung: Der linke Teil der beliebigen Zeichenkette ***x$*** in einer Länge von ***n*** Zeichen wird übergeben.

x$ Beliebiger Zeichenkettenausdruck.

n Beliebiger numerischer Ausdruck, dessen ganzzahliger Wert zwischen 0 und 32767 liegen muß.

Beispiel:

```
Ok
C$="Commodore Bueromaschinen"
PRINT LEFT$(C$,9)
Commodore
Ok
```

Die ersten 9 Zeichen der Zeichenkette C$ werden auf dem Bildschirm ausgegeben.

Anmerkungen:

- Ist ***n*** größer als die Länge von ***x$***, so wird ***x$*** vollständig übergeben.
- Bei ***n*** = 0 wird eine Leer–Zeichenkette (Länge 0) übergeben.
- Weitere teilkettenbildende Funktionen sind MID$ und RIGHT$ (s. dort).

LEN–Funktion

Format: ***v*= LEN(*x$*)**

Bedeutung: Es wird die Länge der Zeichenkette ***x$*** übergeben.

x$ Beliebiger Zeichenkettenausdruck

Beispiel:

```
Ok
C$="Commodore Bueromaschinen"
PRINT LEN(C$)
 24
Ok
```

C$ ist 24 Zeichen lang. Leerzeichen sowie alle nicht druckbaren Zeichen werden mitgezählt.

LET–Anweisung

Format: [**LET**] ***Var= Ausdruck***

Bedeutung: Der Wert eines beliebigen Ausdrucks wird einer Variablen zugewiesen.

Var Jede beliebige einfache Variable oder jedes beliebige Feldelement.

Ausdruck Jeder beliebige Amiga Basic–Ausdruck.

Beispiel:

```
LET A$="Commodore "+"Bueromaschinen"
LET X=A↑2*SQR(210)
```

Anmerkungen:

- Das Schlüsselwort LET kann auch weggelassen werden. Das Gleichheitszeichen (=) ist für eine Zuweisung ausreichend.
- Der Typ des Ausdrucks muß mit dem Variablentyp übereinstimmen; andernfalls wird die Fehlermeldung

  ```
  Type mismatch
  ```

  (keine Typübereinstimmung) angezeigt.

LIBRARY–Anweisung

Format: **LIBRARY "*Dateiname*"**
LIBRARY CLOSE

Bedeutung: Es wird eine Bibliothek mit Maschinensprache–Unterprogrammen für den Zugriff aus Amiga Basic heraus geöffnet.

Dateiname Der Name der Datei, die die benötigten Maschinensprache–Unterprogramme oder –Funktionen enthält.

LIBRARY CLOSE schließt alle geöffneten Bibliotheken.

Beispiel:
```
LIBRARY "graphics.library"
CALL SetDrMd&(WINDOW(8),3)
```

Anmerkungen:

- Es können bis zu 5 Bibliotheken für den Zugriff aus Amiga Basic heraus gleichzeitig geöffnet sein.
- Der Zugriff bleibt solange erlaubt, bis der NEW– oder RUN-Befehl oder die LIBRARY CLOSE–Anweisung gegeben wird.
- Wird die angegebene Bibliothek nicht gefunden, so wird die Fehlermeldung

  ```
  File not found
  ```

  (Datei nicht gefunden) angezeigt. Reicht der Speicher für das Öffnen einer Bibliothek nicht aus, wird die Fehlermeldung

  ```
  Out of memory
  ```

  (Hauptspeicher nicht ausreichend) angezeigt.
- Um die LIBRARY–Anweisung verwenden zu können, muß zuerst eine **.bmap**–Datei auf Diskette erzeugt werden, in der die Unterprogramme der betreffenden Bibliothek beschrieben sind.
- Anhang F enthält ausführliche Informationen über das Erstellen von Bibliotheksdateien.

LINE–Anweisung

Format: **LINE** [[**STEP**] (*x1,y1*)]–[**STEP**] (*x2,y2*)[,[*Farbe*][,B[F]]]

Bedeutung: Es wird eine Linie oder ein Viereck auf dem Bildschirm gezeichnet.

(*x1,y1*),(*x2,y2*) Absolute Bildschirmkoordinaten des Anfangs– und Endpunktes der Linie. Wird der Zusatz **STEP** verwendet, sind die Koordinaten relativ zur augenblicklichen Position des grafischen Cursors (Pixel–Cursor).

Farbe Eine Farbnummer zwischen 0 und 3 entsprechend der mit einer PALETTE–Anweisung (s. dort) gesetzten Farbkennung. Voreingestellt ist die Vordergrundfarbe, die mit der COLOR–Anweisung (s. dort) eingestellt wurde oder die von Amiga Basic standardmäßig verwendet wird (weiß).

B Es wird ein Viereck gezeichnet, dessen gegenüberliegende Ecken durch *x1,y1* und *x2,y2* gegeben sind.

BF Das Viereck wird mit der gegebenen Farbe ausgemalt.

Beispiel 1:

```
LINE -(100,100)
```

Vom letzten angegebenen Punkt wird eine Linie zum Punkt mit den Koordinaten 100,100 in der aktuellen Vordergrundfarbe gezeichnet.

Beispiel 2:

```
LINE (0,0)-(639,199),3
```

Es wird eine Diagonale von oben links nach unten rechts in der Farbe 3 der aktuellen Farbskala gezeichnet.

Beispiel 3:

```
LINE (0,0)-(100,100),,BF
```

Es wird ein Viereck in der aktuellen Vordergrundfarbe gezeichnet und ausgemalt.

Anmerkungen:

- Werden Koordinaten außerhalb des gültigen Bereiches angegeben, so wird der sich daraus ergebende Bildteil an den Grenzen des sichtbaren Bereiches abgeschnitten.

- Nach Ausführung der LINE–Anweisung bezeichnet die Koordinate ***x2,y2*** die aktuelle Position des grafischen Cursors.

LINE INPUT–Anweisung

Format: **LINE INPUT**[;] ["***Text***";]***Zeikettvar***

Bedeutung: Es wird eine Zeichenkette bis zu 255 Zeichen von der Tastatur übernommen und einer angegebene Variablen zugewiesen. Es gibt im Gegensatz zur INPUT–Anweisung kein anderes Trennzeichen als den Wagenrücklaufcode; es werden also alle Zeichen, auch vorlaufende und nachfolgende Leerzeichen, in die Zeichenkette übernommen. Drücken der RETURNTaste schließt die Eingabe ab.

Text Eine beliebige Zeichenkettenkonstante, die vor der Tastatureingabe angezeigt wird und diese z.B. näher erläutert.

Zeikettvar Eine beliebige Zeichenkettenvariable oder ein beliebiges –Feldelement.

Beispiel:

```
LINE INPUT "Namen eingeben: ";A$
PRINT A$
```

Dies ergibt, wenn die geforderten Daten eingegeben werden, z.B:

```
Namen eingeben: Müller, Alfons
Müller, Alfons
Ok
```

Das Komma zwischen Nach– und Vorname gehört zur Eingabe und wird nicht als Trennzeichen erkannt.

Anmerkungen:

- Wird eine numerische Variable angegeben, so meldet der Interpreter den Fehler

  ```
  Type mismatch
  ```

  (keine Typübereinstimmung).

- Im Gegensatz zur INPUT–Anweisung wird vor dem blinkenden Cursor, der zur Eingabe auffordert, kein Fragezeichen (?) ausgegeben.

- Nach dem Drücken der RETURN–Taste ist die aktuelle Cursorposition die Stelle hinter dem zuletzt eingegebenen Zeichen, wenn direkt hinter dem Schlüsselwort LINE INPUT ein Semikolon (;) angegeben wird; andernfalls ist es die erste Schreibposition der nächsten Zeile.

- Die LINE INPUT–Anweisung kann durch Drücken der beiden Tasten Amiga–. oder CTRL–C abgebrochen werden. Der Interpreter kehrt dann auf die Befehlsebene zurück. Ein ggf. dann eingegebener CONT–Befehl setzt das Programm mit der abgebrochenen LINE INPUT–Anweisung fort.

LINE INPUT #–Anweisung

Format: **LINE INPUT #*Dateinr,Zeikettvar***

Bedeutung: Es werden Daten zeilenweise (bis zu 32767 Zeichen) aus einer sequentiellen Datei gelesen und einer Zeichenkettenvariablen zugewiesen. Es gibt im Gegensatz zu der Anweisung INPUT # keine Trennzeichen (Komma oder Leerstellen). Der Wagenrücklaufcode beendet den Lesevorgang.

Dateinr Die Nummer, unter der die Eingabedatei eröffnet wurde.

Zeikettvar Eine beliebige Zeichenkettenvariable oder ein beliebiges –Feldelement.

Beispiel:

```
OPEN "NAMEN" FOR INPUT AS #3
LINE INPUT #3,A$
PRINT A$:CLOSE 3
```

Es wird eine Zeichenkette aus der Datei NAMEN gelesen und angezeigt.

Anmerkungen:

- Mit der LINE INPUT #–Anweisung werden grundsätzlich alle Zeichen bis zum Wagenrücklaufcode übernommen. Bei der nächsten LINE INPUT #–Anweisung werden der Wagenrücklauf– sowie ein ggf. folgender Zeilenvorschubcode der vorher gelesenen Zeichenkette übergangen, und es wird bis zum nächsten Wagenrücklaufcode gelesen.

LIST–Befehl

Format: **LIST** [***Zeile1***][–[***Zeile2***]][,***Dateiang***]

Bedeutung: Das aktuelle, sich im Hauptspeicher befindende Amiga Basic-Programm wird im List–Fenster oder einer anderen spezifizierten Ausgabeeinheit gelistet.

Zeile1,Zeile2 Gültige Zeilennummern im Bereich 0 bis 65529 oder alphanumerische Sprungmarken. Es wird immer einschließlich ***Zeile1*** und ***Zeile2*** gelistet.

Dateiang Ein Zeichenkettenausdruck für eine Dateispezifikation, wie in Kapitel 5.2 Wird die Dateiangabe weggelassen, so wird der angegeben Zeilenbereich im List–Fenster gelistet.

Beispiel 1:

```
LIST
```

Das gesamte Programm wird im List–Fenster gelistet.

Beispiel 2:

```
LIST -100,"SCRN:"
```

Alle Zeilen bis Zeile 100 einschließlich werden im List–Fenster gelistet.

Beispiel 3:

```
LIST ,"LPT1:"
```

Ausdruck des gesamten Programms auf dem Drucker.

Beispiel 4:

```
LIST 1000-,"COM1:2400,N,8"
```

Alle Zeilen des Programms ab Zeile 1000 werden über den Datenfernübertragungsanschluß mit 2400 Baud ohne Parität mit 8 Daten–Bits und einem Stop–Bit übertragen (s.a. Kapitel 5.1).

Anmerkungen:

- Die Ausgabe der Programmliste in eine Disk–Datei erfolgt im ASCII–Format.
- Nach der Ausführung von LIST kehrt der Interpreter grundsätzlich zur Befehlsebene zurück.
- Kapitel 3.3 und 4.1 enthalten weitere Hinweise zur Verwendung des List–Fensters.

LLIST–Befehl

Format: **LLIST** [*Zeile1*][–[*Zeile2*]]

Bedeutung: Der angegebene Zeilenbereich oder das gesamte im Hauptspeicher befindliche Amiga Basic–Programm wird auf dem Drucker (PRT:) gelistet. Für die Zeilenangaben gelten dieselben Regeln wie beim LIST–Befehl (s. dort).

Beispiele:

```
LLIST
```

Listet das gesamte Programm

```
LLIST -100
```

Listet alle Programmzeilen bis einschließlich Zeile 100.

```
LLIST 100-200
```

Listet alle Programmzeilen zwischen den Zeilen 100 und 200 einschließlich.

```
LLIST 1000-
```

Listet alle Programmzeilen ab einschließlich Zeile 1000.

Anmerkungen:

- Nach der Ausführung von LLIST kehrt der Interpreter grundsätzlich zur Befehlsebene zurück.
- Die Ausgabe erfolgt grundsätzlich auf dem Drucker PRT: (s.a. AmigaDOS–Benutzerhandbuch).

LOAD–Befehl

Format: **LOAD** [***Dateiangabe***[,**R**]]

Bedeutung: Es wird ein Programm von einer angegebenen Eingabeeinheit in den Hauptspeicher geladen und ggf. gestartet.

Dateiangabe Ein Zeichenkettenausdruck für eine Dateispezifikation, wie in Kapitel 5.2 beschrieben.

R Wird der Parameter **R** angegeben, so wird das geladene Programm gestartet. Andernfalls kehrt der Interpreter nach dem Ladevorgang zur Befehlsebene zurück.

Beispiel 1:

```
LOAD "PROG1"
```

Das Programm PROG1 wird von der Diskette im aktuellen Laufwerk geladen.

Beispiel 2:

```
LOAD "DF1:PROG2",R
```

Das Programm PROG2 wird von der Diskette in Laufwerk DF1: geladen und gestartet.

Anmerkungen:

- Wird keine Dateiangabe gegeben, so wird der Anwender mit einem Kommunikationsfenster zur Eingabe des gewünschten Dateinamens aufgefordert.
- Die Dateiangabe muß den Dateinamen enthalten, unter dem das Programm ursprünglich mit SAVE (s. dort) gespeichert wurde.
- Durch den Befehl LOAD ohne Angabe von **R** werden vor dem Laden alle ggf. geöffneten Dateien geschlossen, und das ggf. vorher im Speicher befindliche Programm wird gelöscht.
- Beim LOAD–Befehl mit Angabe von **R** bleiben alle ggf. vorher geöffneten Dateien geöffnet. Auf diese Weise können Programme verkettet werden, wobei allerdings die Datenübergabe über Diskettendateien erfolgen muß (s.a. RUN–Befehl).
- Siehe auch CHAIN–Anweisung sowie MERGE– und SAVE–Befehl.

LOC–Funktion

Format: ***v*= LOC(*Dateinr*)**

Bedeutung: Bei Direktzugriffsdateien liefert die LOC–Funktion die Nummer des zuletzt gelesenen oder geschriebenen physikalischen Datenblocks.

Bei sequentiellen Dateien liefert die LOC–Funktion die Anzahl der bereits geschriebenen oder gelesenen Bytes, dividiert durch die voreingestellte (128) oder gewählte Satzlänge.

Dateinr Die Nummer, unter der die bezogene Datei eröffnet wurde.

Beispiel 1:

```
IF LOC(3)>10 THEN 500
```

Wird nach dem 10. Datenblock noch ein Block gelesen (oder geschrieben), verzweigt das Programm nach Zeile 500.

Beispiel 2:

```
PUT #2,LOC(2)
```

Der zuletzt gelesene Satz wird überschrieben.

Anmerkungen:

- Bei sequentiellen Dateien liefert LOC vor dem ersten Lesen eine 1, da beim Eröffnen der Datei bereits der erste Datenblock gelesen wird.
- Wird eine Datei für KYBD: (Tastatur) oder COM1: (serielle Schnittstelle) geöffnet und darauf die LOC–Funktion angewendet, so erhält man den Wert 1, wenn irgendwelche Daten zum Auslesen bereitstehen, andernfalls 0.

LOCATE–Anweisung

Format: **LOCATE** [***Zeile***][,***Spalte***]

Bedeutung: Positioniert den Textcursor im aktuellen Ausgabefenster.

Zeile Ein ganzzahliger Ausdruck, dessen Wert größer als 0 sein muß, und der die Zeile angibt, in der der Cursor positioniert werden soll.

Spalte Ein ganzzahliger Ausdruck, dessen Wert größer als 0 sein muß, und der die Spalte angibt, in der der Cursor positioniert werden soll.

Beispiel:

```
Y=CSRLIN 'merkt sich vertikale Position
X=POS(0) 'merkt sich horizontale Position
LOCATE 20,1
PRINT "Dies ist Zeile 20 (Schirm-Ende)"
LOCATE Y,X
PRINT "Dies war die ursprüngliche Cursor-Position"
```

Bei diesem Beispiel wird die alte Cursor–Position gespeichert, dann am unteren Schirmende eine Meldung angezeigt und schließlich eine weitere Meldung bei der ursprünglichen Cursor–Position angezeigt.

Anmerlungen:

- Zeilen– und Spaltenangaben müssen für das bezogene Ausgabefenster gültige Werte haben.
- Zur Bestimmung der exakten Bildschirmposition benutzt Amiga Basic die Höhe und Breite der Ziffer Null (0) des für das aktuelle Ausgabefenster gewählten Zeichensatzes.
- Werden Zeilen– und Spalten–Parameter weggelassen, so nimmt LOCATE die aktuelle Position des Text–Cursors.

LOF–Funktion

Format: ***v*= LOF(*Dateinr*)**

Bedeutung: Es wird die Länge einer Datei in Bytes übergeben.

Dateinr Nummer, unter der die Datei eröffnet wurde.

Beispiel:

```
OPEN "ADRESSEN" AS #3 LEN=512
PRINT LOF(3)/512
```

Es wird die Anzahl von 512–Byte langen Datenblöcken in der Direktzugriffsdatei ADRESSEN angezeigt.

Anmerkungen:

- Bei Dateien, die für SCRN: (Bildschirm), KYBD: (Tastatur), LPT1: (Drucker) oder COM1: (serielle Schnittstelle) geöffnet wurden, liefert LOF immer den Wert 0.

LOG–Funktion

Format: ***v*=LOG(*x*)**

Bedeutung: Es wird der natürliche Logarithmus eines beliebigen numerischen Ausdruckes größer als 0 berechnet. Ist das Argument einfach genau, erfolgt die Berechnung auch in einfacher Genauigkeit. Ist es doppelt–genau, wird auch ein doppelt–genauer Wert übergeben.

x Beliebiger numerischer Ausdruck, dessen Wert größer als Null sein muß.

Beispiel 1:

```
PRINT LOG(2.718282)
 1
Ok
```

Der natürliche Logarithmus der Zahl e ist 1.

Beispiel 2:

```
PRINT LOG(-1)
```

führt zur Fehleranzeige

```
Illegal function call
```

(Unerlaubter Funktionsaufruf).

Der natürliche Logarithmus ist nur für positive Zahlen definiert.

LPOS–Funktion

Format: ***v*=LPOS(*n*)**

Bedeutung: Es wird die Position des zuletzt gedruckten Zeichens im Drukkerpuffer übergeben.

n ist ein Scheinargument ohne Bedeutung, das jedoch angegeben werden muß.

Beispiel:

```
IF LPOS(X)>80 THEN LPRINT CHR$(13)
```

Spätestens nach dem 80. Zeichen soll ein Wagenrücklaufcode gedruckt werden.

Anmerkungen:

- Der von LPOS gelieferte Wert muß nicht zwingend mit der physikalischen Druckkopfposition im Drucker übereinstimmen.

LPRINT– und LPRINT USING–Anweisungen

Format: **LPRINT** [***Liste von Ausdr***][;]
LPRINT USING ***v$***;***Liste von Ausdr***[;]

Bedeutung: Es werden Daten auf dem Drucker LPT1: ausgegeben.

Liste von Ausdr Eine beliebige Liste von numerischen und/ oder Zeichenkettenausdrücken, deren Werte gedruckt werden sollen. Die einzelnen Ausdrücke müssen durch Komma (,) oder Semikolon (;) getrennt sein, wenn der vorangegangene Ausdruck nicht mit einem der Zeichen) $ # % & " endet. Die Liste muß zusammen mit dem Schlüsselwort LPRINT oder LPRINT USING in einer Programmzeile stehen.

v$ Zeichenkettenvariable oder –konstante, die das Druckformat bestimmt (s. dazu PRINT USING–Anweisung).

Beispiel: Da LPRINT und LPRINT USING genauso wie PRINT und PRINT USING arbeiten, wird auf die dort angeführten Beispiele verwiesen.

Anmerkungen:

- Es wird eine Druckbreite von 80 Zeichen vorausgesetzt, so daß nach dem Drucken des 80. Zeichens vom Interpreter automatisch ein Wagenrücklauf/Zeilenvorschub an den Drucker ausgegeben wird. Beim Drucken von exakt 80 Zeichen langen Zeilen sollte die zu druckende Liste von Ausdrücken deshalb mit einem Semikolon (;) in der Anweisung abgeschlossen werden. Die Druckbreite kann mit der Anweisung WIDTH "LPT1:" oder WIDTH LPRINT (s. dort) geändert werden.

LSET–Anweisung

Format: **LSET *Zeichenkettenvariable = x$***

Bedeutung: Überträgt in Vorbereitung für die PUT–Anweisung für Dateien Daten linksbündig in einen Dateipuffer für wahlfreien Zugriff (Direktzugriffsdatei).

Zeichenkettenvariable Der Name einer Variablen, die in einer FIELD–Anweisung als Datenfeld definiert wurde.

x$ Beliebiger Zeichenkettenausdruck, dessen Wert die Daten repräsentiert, die in dem durch die Zeichenkettenvariable bestimmten Feld linksbündig abgelegt werden sollen.

Beispiel:

```
LSET Y$=MKS$(NUM)
```

Der numerische Wert NUM wird bei der linksbündigen Speicherung im Datenfeld Y$ als Zeichenkette interpretiert (s. Anmerkungen und MKS$–Funktion).

Anmerkungen:

- Ist die zu speichernde Datenlänge kleiner als das definierte Feld, so wird das Feld rechts mit Leerstellen aufgefüllt. Ist sie größer, wird rechts abgeschnitten.
- Numerische Werte müssen vor der Speicherung mit LSET oder RSET (s. dort) durch die Funktionen MKI$, MKL$, MKS$ oder MKD$ (s. dort) als Zeichenketten interpretiert und behandelt werden.
- In LSET– und RSET–Anweisungen können auch durch andere als durch FIELD–Anweisungen definierte Datenfelder verwendet werden (z.B. für Druckformat–Erstellung).
- Weitere Informationen zum Datenaustausch mit Dateien für wahlfreien Zugriff sind in Kapitel 5.4 zu finden.

MENU–Anweisung

Format: **MENU** ***Kennung,Punkt,Status***[**,** ***Titel***]
MENU RESET

Bedeutung: Erzeugt ein Anwender–Menü in der Menü–Leiste mit Menü-Punkten oder setzt die Menü–Leiste auf ihren Originalzustand zurück.

Kennung Ein ganzzahliger Ausdruck, dessen Wert zwischen 1 und 10 liegen muß, und der die Nummer des Menütitels in der Menü–Leiste angibt.

Punkt Ein ganzzahliger Ausdruck, dessen Wert zwischen 0 und 19 liegen muß, und der die Nummer des Menü–Punktes unterhalb des Titels in der Menü–Leiste angibt. Der Wert 0 spezifiziert einen Menü–Titel.

Status Ein ganzzahliger Ausdruck, dessen Wert zwischen 0 und 2 liegen muß. Ein Wert von 0 inaktiviert je nach Wert von ***Punkt*** das gesamte Menü oder den spezifizierten Menü–Punkt. Ein Wert von 1 aktiviert das gesamte Menü oder den spezifizierten Menü–Punkt. Ein Wert von 2 aktiviert den spezifizierten Menüpunkt und versieht ihn mit einer entsprechenden Kennzeichnung. Soll ein Menü–Punkt mit Kennzeichnung versehen werden, so müssen bei seinem Titel zwei Leerstellen vor dem Text vorgesehen werden.

Titel Ein Zeichenkettenausdruck oder eine Zeichenkettenkonstante in Anführungszeichen ("), dessen/deren Wert den Titel für das Menü in der Menü–Leiste oder die Bezeichnung des Menüpunktes darstellt.

Beispiel:

```
MENU 1,0,1,"Kontenbewegungen"
MENU 1,1,1,"Einzahlungen"
MENU 1,2,1,"Entnahmen"
MENU 1,3,1,"Dauerauftraege"
MENU 1,4,1,"Kreditkarten-Kauf"
```

Diese Anweisungen richten in der Menü–Leiste an erster Position das Menü "Kontenbewegungen" mit 4 Menü–Punkten ein und aktivieren es.

Anmerkungen:

- Wenn ein Status von 1 bei einer Menü–Titelspezifikation angegeben wird, so wird dieser Titel anstelle des vorher dort befindlichen Titels in der Titelleiste angezeigt.

MENU–Funktion

Format: ***v*= MENU(*n*)**

Bedeutung: Liefert die Nummer des zuletzt gewählten Menü–Titels oder –Punktes.

n Ein ganzzahliger Ausdruck, dessen Wert 0 oder 1 sein darf und der folgende Bedeutung hat:

0 Es wird die Nummer des zuletzt gewählten Menü–Titels übergeben. MENU(0) wird nach jeder Ausführung zurückgesetzt, so daß die Menü–Leiste so abgefragt werden kann wie die Tastatur mit der INKEY$–Funktion.

1 Es wird die Nummer des zuletzt gewählten Menü–Punktes übergeben.

Beispiel:

```
Menu=MENU(0)
MenuPunkt=MENU(1)
```

Anmerkungen:

- Weitere Anweisungen im Zusammenhang mit der Arbeit mit Menüs sind MENU, MENU ON, ON MENU und SLEEP (s. dort).

MENU ON/OFF/STOP–Anweisungen

Formate: **MENU ON**
MENU OFF
MENU STOP

Bedeutung: Die Unterbrechungsreaktionsfähigkeit bei Menü–Benutzung durch den Anwender wird aktiviert oder inaktiviert.

Beispiel:
```
ON MENU GOSUB TestMenu
ON MOUSE GOSUB TestMaus
MENU ON
MOUSE ON
```

Anmerkungen:

- Die Anweisung MENU ON muß ausgeführt werden, um eine Unterbrechungsverzweigung bei Menu–Benutzung durch den Anwender zu erreichen.
- Nach MENU OFF kann bei Menü–Benutzung nicht mehr programmiert verzweigt werden.
- Nach MENU STOP speichert Amiga Basic eine Menu–Benutzung durch den Anwender, unterbricht das Programm mit einer Verzweigung aber erst, sobald eine MENU ON–Anweisung gegeben wird.
- Weitere Hinweise finden Sie bei den MENU– und ON MENU–Anweisungen sowie im Kapitel 6.4.

MERGE–Befehl

Format: **MERGE** ***Dateiangabe***

Bedeutung: Es wird eine Amiga Basic–Programmdatei im ASCII–Format an ein im Hauptspeicher befindliches Proramm angefügt.

Dateiangabe Ein Zeichenkettenausdruck für eine Dateispezifikation, wie in Kapitel 5.2 beschrieben.

Beispiel:

```
MERGE "PROG2"
Ok
```

Das Programm PROG2 auf der Diskette im aktuellen Laufwerk wird an das hauptspeicherresidente Programm angefügt.

Anmerkungen:

- Nach MERGE kehrt Amiga Basic grundsätzlich auf die Befehlsebene in den Direktmodus zurück.
- Wurde das anzufügende Programm nicht im ASCII–Format gespeichert (s. SAVE–Befehl), so wird der Mischvorgang mit der Fehlermeldung

  ```
  Bad file mode
  ```

  (falscher Dateityp) abgebrochen und das hauptspeicherresidente Programm wird nicht verändert.

MID$–Anweisung

Format: **MID$(*v$*,*n*[,*m*]) = *x$***

Bedeutung: Es wird ein Teil einer Zeichenkette durch eine andere Zeichenkette ersetzt.

v$ Die Zeichenkettenvariable oder das –feldelement, von der/dem ein Teil ersetzt werden soll.

n Ein ganzzahliger Ausdruck mit einem Wert zwischen 1 und 32767, der die Position angibt, ab der ersetzt werden soll.

m Ein ganzzahliger Ausdruck mit einem Wert zwischen 0 und 32767, der, wird er angegeben, die Anzahl der zu ersetzenden Zeichen bestimmt. Andernfalls wird ab Position ***n*** mit dem gesamten ***x$*** ersetzt.

x$*** Ein beliebiger Zeichenkettenausdruck, dessen Wert die Daten darstellt, die einen Teil von ***v$ ersetzen sollen.

Beispiel:

```
A$="Commodore Bueromaschinen"
MID$(A$,11)="Deutschland  "
PRINT A$
Commodore Deutschland
Ok
```

Anmerkungen:

- Grundsätzlich wird die Länge von ***v$*** nicht verändert, wenn ersetzt wird, d.h. es werden maximal so viele Zeichen ersetzt, wie LEN(***v$***)–***n*** + 1 angibt.

- Liegen ***n*** und ***m*** außerhalb des angegebenen Bereiches, wird die Fehlermeldung

  ```
  Illegal function call
  ```

  (unerlaubter Funktionsaufruf) angezeigt.

MID$–Funktion

Format: ***v$*** **= MID$(*x$,n*[,*m*])**

Bedeutung: Es wird ein spezifizierter Teil einer Zeichenkette übergeben.

x$ Ein beliebiger Zeichenkettenausdruck, von dessen Wert ein Teil übergeben werden soll.

n Ein ganzzahliger Ausdruck zwischen 1 und 32767, der die Position angibt, an der die Teilzeichenkette beginnt.

m Ein ganzzahliger Ausdruck zwischen 0 und 32767, der, wird er angegeben, die Länge der zu bildenden Teilzeichenkette angibt. Wird ***m*** weggelassen oder existieren von Position ***n*** an weniger als ***m*** Zeichen, werden von ***n*** an alle Zeichen von ***x$*** übergeben. Ist ***m*** gleich Null, oder ist ***n*** größer als LEN(***x$***), so wird eine Leer–Zeichenkette (Länge Null) übergeben.

Beispiel:

```
A$="Heute ist"
B$="SonntagMontagDienstag"
PRINT A$;MID$(B$,8,6)
Heute ist Montag
Ok
```

Anmerkungen:

- Liegen ***n*** und ***m*** außerhalb des angegebenen Bereiches, wird der Fehler

  ```
  Illegal function call
  ```

  (unerlaubter Funktionsaufruf) angezeigt.

MKI$-, MKL$-, MKS$-, MKD$–Funktionen

Formate:

***v$*** **= MKI$(*kurz–ganzzahliger Ausdruck*)**
***v$*** **= MKL$(*lang–ganzzahliger Ausdruck*)**
***v$*** **= MKS$(*einfachgenauer Ausdruck*)**
***v$*** **= MKD$(*doppeltgenauer Ausdruck*)**

Bedeutung: Es werden die Werte numerischer Ausdrücke unterschiedlicher Genauigkeit in ihrer internen Darstellung (s. Anhang E) als Zeichenketten interpretiert.

Beispiel:

```
OPEN "B:LAGER" AS #3
FIELD #3,2 AS ARTNR$,20 AS BEZ$,2 AS STZ$
FOR I=1 TO 100
    LSET ARTNR$=MKI$(AN%(I))
    LSET BEZ$=BZ$(I)
    LSET STZ$=SZ$(I)
    PUT #3,I
NEXT
```

In diesem Beispiel werden nach der Eröffnung einer Datei für wahlfreien Zugriff jedem Datensatz drei Variablen unterschiedlicher Länge zugeordnet. In einer Schleife werden die Werte in den Puffer übertragen, wobei die Artikelnummer aus einem Ganzzahlfeld AN%(I) mit Hilfe der Funktion MKI$ als Zeichenkette interpretiert, in den Datensatz eingefügt und dann der Datensatz auf Disk gespeichert wird.

Anmerkungen:

- Soll der Wert eines numerischen Ausdruckes in einen Dateipuffer für wahlfreien Zugriff mit Hilfe der LSET– oder RSET-Anweisung gebracht werden, so muß er dazu vorher entsprechend seiner Genauigkeit mit einer der hier beschriebenen Funktionen als Zeichenkette umdefiniert werden, die, ebenfalls abhängig von der Genauigkeit, folgende Länge benötigt:

  MKI$ 2 Bytes
  MKL$ 4 Bytes
  MKS$ 4 Bytes
  MKD$ 8 Bytes

- Der Unterschied zur STR$–Funktion besteht darin, daß hier die Datenbytes nicht verändert, sondern nur anders interpretiert werden (s.a. CVI–, CVL–, CVS–, CVD–Funktionen, LSET– und RSET–Anweisungen sowie Kapitel 5.4). Die MKx$–Funktionen liefern also Zeichenketten, die die angegebenen numerischen Werte in der internen binären Darstellung (Ganzzahl oder Gleitpunkt) repräsentieren.

MOUSE–Funktion

Format: ***v*= MOUSE(*n*)**

Bedeutung: Die MOUSE–Funktion liefert Informationen über den Status der linken Maus–Taste (Auswahltaste) sowie über den Maus–Zeiger (Pfeil) innerhalb des aktiven Fensters.

n Ein ganzzahliger Ausdruck, dessen Wer t zwischen 0 und 6 liegen muß und der die von der Funktion gelieferte Information bestimmt:

0 Liefert den Status der linken Maus–Taste. Nach der Ausführung von MOUSE(0) speichert Amiga Basic Start– und End–Positionen der Maus, bis eine weitere MOUSE(0)-Funktion ausgeführt wird. Deshalb sollte ein Programm nach der Wahrnehmung einer Mausbewegung durch MOUSE(0) mit Hilfe der Funktionen MOUSE(3), MOUSE(4), MOUSE(5) und MOUSE(6) (s. unten) Start– und Endpositionen der Maus feststellen. MOUSE(0) liefert folgende Werte:

0 Die linke Maustaste ist gegenwärtig nicht gedrückt und wurde auch seit dem letzten MOUSE(0)–Funktionsaufruf nicht gedrückt.

1 Die linke Maustaste ist gegenwärtig nicht gedrückt, wurde jedoch seit dem letzten MOUSE(0)–Funktionsaufruf einmal gedrückt. Zur Bestimmung der Start– und Endpositionen der Maus während der Auswahl sind die Funktionen MOUSE(3), MOUSE(4), MOUSE(5) und MOUSE(6) zu verwenden.

2 Die linke Maustaste ist gegenwärtig nicht gedrückt, wurde jedoch seit dem letzten MOUSE(0)–Funktionsaufruf zweimal gedrückt. Zur Bestimmung der Start– und Endpositionen der Maus während der Auswahl sind die Funktionen MOUSE(3), MOUSE(4), MOUSE(5) und MOUSE(6) zu verwenden. (Entsprechend bedeutet ein Wert von 3, daß die Auswahltaste dreimal gedrückt wurde usw.).

–1 Der Anwender hält die linke Maustaste nieder, nachdem er sie einmal gedrückt hatte. Dieser Wert bedeutet meistens, daß die Maus gegenwärtig bewegt wird.

–2 Der Anwender hält die linke Maustaste nieder, nachdem er sie zweimal gedrückt hatte. Dieser Wert bedeutet meistens, daß die Maus gegenwärtig bewegt wird. (Entsprechend bedeutet ein Wert von –3, daß die Auswahltaste vor dem Niederhalten dreimal gedrückt wurde usw.).

1 MOUSE(1) liefert die horizontale x–Koordinate zum Zeitpunkt des letzten MOUSE(0)–Funktionsaufrufes, und zwar unabhängig vom Status der Auswahltaste.

2 MOUSE(2) liefert die vertikale y–Koordinate zum Zeitpunkt des letzten MOUSE(0)–Funktionsaufrufes, und zwar unabhängig vom Status der Auswahltaste.

3 MOUSE(3) liefert die horizontale x–Koordinate zum Zeitpunkt der letzten Betätigung der Auswahltaste vor einem MOUSE(0)–Funktionsaufruf. MOUSE(3) in Verbindung mit MOUSE(4) (s. unten) dient zur Bestimmung der Startkoordinaten einer Mausbewegung.

4 MOUSE(4) liefert die vertikale y–Koordinate zum Zeitpunkt der letzten Betätigung der Auswahltaste vor einem MOUSE(0)-Funktionsaufruf.

5 Wenn die linke Maustaste nach dem letzten MOUSE(0)-Funktionsaufruf gedrückt wurde, liefert MOUSE(5) die horizontale x–Koordinate der Position, bei der der Maus-Cursor war, als MOUSE(0) aufgerufen wurde. War die linke Maustaste dagegen nach dem letzten MOUSE(0)–Funktionsaufruf nicht gedrückt, liefert MOUSE(5) die horizontale x–Koordinate der Position, bei der die linke Maustaste losgelassen wurde. Mit der MOUSE(5)–Funktion kann die Mausbewegung verfolgt werden und die Position bestimmt werden, bei der die Bewegung aufgehört hat.

6 MOUSE(6) arbeitet genau wie MOUSE(5), liefert nur jeweils die vertikale y–Koordinate.

Beispiel:

```
CheckMouse:
    IF MOUSE(0)=0 THEN CheckMouse
    IF ABS(X-MOUSE(1)) > 2 THEN MovePicture
    IF ABS(Y-MOUSE(2)) < 3 THEN CheckMouse
MovePicture:
    PUT (X,Y),P
    X=MOUSE(1):Y=MOUSE(2)
    PUT (X,Y),P
    GOTO CheckMouse
```

Bei diesem Beispiel wird die Mausbewegung überwacht. Wenn die Maus bei gedrückter Auswahltaste bewegt wird, wird ein grafisches Objekt, das im Feld P gespeichert ist, in eine neue Position bewegt.

Anmerkungen:

- Mit der MOUSE–Funktion kann **nicht** die rechte Maustaste (Menü–Taste) überwacht werden. Dazu dient vielmehr die MENU–Funktion (s. dort)

MOUSE ON/OFF/STOP–Anweisungen

Formate: **MOUSE ON**
MOUSE OFF
MOUSE STOP

Bedeutung: Die Unterbrechungsreaktionsfähigkeit bei Betätigung der Auswahltaste (linke Taste) der Maus durch den Anwender wird aktiviert oder inaktiviert.

Beispiel: Siehe MENU ON/OFF/STOP–Anweisungen.

Anmerkungen:

- Die Anweisung MOUSE ON muß ausgeführt werden, um eine Unterbrechungsverzweigung bei Betätigung der Auswahltaste der Maus durch den Anwender zu erreichen.
- Nach MOUSE OFF kann bei Auswahltastenbetätigung nicht mehr programmiert verzweigt werden.
- Nach MOUSE STOP speichert Amiga Basic eine Auswahltastenbetätigung durch den Anwender, unterbricht das Programm mit einer Verzweigung aber erst, sobald eine MOUSE ON–Anweisung gegeben wird.
- Weitere Hinweise finden Sie bei der MOUSE–Funktion und ON MOUSE–Anweisung sowie im Kapitel 6.4.

NAME–Befehl

Format: **NAME *Dateiangabe* AS *Dateiname***

Bedeutung: Es wird einer Disketten– oder Festplattendatei ein neuer Name zugewiesen.

Dateiangabe Ein Zeichenkettenausdruck für eine Dateispezifikation, wie in Kapitel 5.2 beschrieben.

Dateiname Ein Zeichenkettenausdruck für einen gültigen Dateinamen, wie in Kapitel 5.2 beschrieben.

Beispiel: **NAME "ADRESSEN.TXT" AS "ANSCHR.TXT"**

Die Datei ADRESSEN.TXT auf der Diskette im aktuellen Laufwerk erhält den Namen ANSCHR.TXT.

Anmerkungen:

- Es wird eine Fehlermeldung angezeigt, wenn die umzubenennende Datei auf dem spezifizierten Laufwerk nicht existiert oder wenn der neue Name auf diesem Laufwerk bereits für eine andere Datei vergeben ist.

- Fehlt die Angabe eines Laufwerks (Diskette oder Festplatte), so wird das im AmigaDOS voreingestellte Laufwerk angenommen.

NEW–Befehl

Format: **NEW**

Bedeutung: Das im Hauptspeicher befindliche Programm wird zusammen mit all seinen Variablen und dem List–Fenster gelöscht.

Beispiel:

```
a$="Amiga":PRINT a$
Commodore PC
Ok
NEW
Ok
PRINT a$
Ok
```

Anmerkungen:

- Nach der Ausführung von NEW kehrt der Interpreter immer in die Befehlsebene (Direktmodus) zurück, nachdem das Ausgabefenster gelöscht wurde.
- Mit der Ausführung von NEW werden alle geöffneten Dateien geschlossen sowie eine ggf. eingeschaltete Programmablaufüberwachung (s. TRON–Befehl) ausgeschaltet.
- NEW kann auch aus dem **Project**–Menü gewählt werden.
- Befindet sich zum Zeitpunkt der Ausführung des NEW–Befehls ein Programm im Speicher, daß nach dem Laden verändert wurde oder noch nicht gespeichert wurde, so wird der Anwender in einem Komunikationsfenster aufgefordert, zu entscheiden, ob er dieses Programm vor der Ausführung von NEW auf Disk speichern will oder nicht.
- NEW verändert nicht die Position oder Größe von Fenstern.

NEXT–Anweisung

Siehe FOR–Anweisung.

OBJECT.AX– und OBJECT.AY–Anweisungen

Format: **OBJECT.AX** ***Objekt,Wert***
OBJECT.AY ***Objekt,Wert***

Bedeutung: Definiert die Beschleunigung eines grafischen Objektes (Bob oder Sprite) in X– bzw. Y–Richtung

Objekt Ein ganzzahliger Ausdruck, dessen Wert größer als 0 sein muß, und der der Objekt-Kennung in der OBJECT.SHAPE-Anweisung (s. dort) entspricht.

Wert Einganzzahliger Ausdruck, dessen Wert größer als 1 sein muß, und der die Beschleunigungsrate in Bildpunkten pro Sekunde angibt.

Beispiel: Siehe OBJECT.SHAPE–Anweisung.

OBJECT.CLIP–Anweisung

Format: **OBJECT.CLIP (*x1,y1*)–(*x2,y2*)**

Bedeutung: Definiert ein Rechteck, dessen Grenzen die Begrenzung beim Zeichnen eines Objektes (Bob oder Sprite) darstellen. Voreingestellt ist hier das aktuelle Ausgabe–Fenster.

x1,y1 Koordinaten der oberen linken Ecke des Rechtecks.

x2,y2 Koordinaten der unteren rechten Ecke des Rechtecks.

Anmerkungen:

- In Kapitel 7 wird beschrieben, wie Objekte mit Hilfe des Objekt–Editors erzeugt werden können. Der Objekt–Editor ist ein Amiga Basic–Programm.
- Wenn Sie die Größe des Ausgabefensters durch Ziehen des Größen–Symbols mit der Maus verändern, werden die Grenzen des Objektbereiches, wie sie mit der letzten OBJECT.CLIP-Anweisung definiert werden, **nicht** verändert.

OBJECT.CLOSE–Anweisung

Format: **OBJECT.CLOSE** [***Objekt***[,***Objekt***. . .]]

Bedeutung: Gibt den von einem, mehreren oder allen Objekten belegten Speicherplatz wieder frei, wenn die angegebenen Objekte nicht mehr benötigt werden.

Objekt Ein ganzzahliger Ausdruck, dessen Wert größer als 0 sein muß, und der der Objekt–Kennung in der OBJECT.SHAPE-Anweisung (s. dort) entspricht. ***Objekt*** bezeichnet das Objekt im aktuellen Ausgabefenster, das nicht mehr benötigt wird. Wenn keine Objekt–Kennungen angegeben werden, wird der für alle Objekte des aktuellen Ausgabefensters reservierte Speicherplatz wieder freigegeben.

OBJECT.HIT–Anweisung

Format: **OBJECT HIT** ***Objekt***[,***Selbst***][,***Fremd***]

Bedeutung: Legt fest, ob bei einer Kollision von ***Objekt*** mit der Fensterbegrenzung oder mit einem anderen Objekt ein Unterbrechungsereignis, das mit der ON COLLISION GOSUB–Anweisung (s. dort) programmiert verarbeitet werden kann, stattfinden soll oder nicht.

Objekt Ein ganzzahliger Ausdruck, dessen Wert größer als 0 sein muß, und der der Objekt–Kennung in der OBJECT.SHAPE–Anweisung (s. dort) entspricht.

Selbst Ein ganzzahliger Wert zwischen 1 und 65535, der das Dezimaläquivalent einer 16–Bit–Maske darstellt. Jedes Bit dieser Maske korrespondiert zu dem entsprechenden Bit der ***Stoß***-Maske (s. unten).

Stoß Ein ganzzahliger Wert zwischen 0 und 65535, der das Dezimaläquivalent einer 16–Bit–Maske darstellt. Ist das niederwertigste Bit dieser Maske gesetzt, so bedeutet dies, daß bei einer Kollision des zugehörigen ***Objekt***s mit der Fensterbegrenzung ein COLLISION–Unterbrechungsereignis eintritt, das mit einer ON COLLISION GOSUB–Anweisung (s. dort) verarbeitet werden kann. Jedes andere gesetzte Bit dieser Maske bedeutet, daß bei einer Kollision dieses ***Objekt***s mit einem anderen Objekt, in dessen ***Selbst***–Maske an der entsprechenden Stelle ebenfalls ein Bit gesetzt ist, ein COLLISION–Unterbrechungsereignis eintritt.

Kollidiert dagegen ***Objekt*** mit einem anderen Objekt, wobei die logische UND–Verknüpfung von ***Selbst*** mit der Stoß–Maske des anderen Objektes den Wert Null ergibt, so tritt **kein** Unterbrechungsereignis ein; die beiden Objekte können sich also ohne Reaktion gegenseitig durchdringen.

Beispiel:

```
OBJECT.SHAPE 1,Asteroid$
OBJECT.SHAPE 2,Schiff$
OBJECT:SHAPE 3,Geschoss$
OBJECT.HIT 1,8,7'kollidiert mit Rand, Schiff,
Geschoss
OBJECT.HIT 2,2,9'kollidiert mit Rand, Asteroid
OBJECT.HIT 3,4,9'kollidiert mit Rand, Asteroid
```

Anmerkungen:

- Voreinstellungsmäßig kollidieren alle Objekte miteinander und mit der Fensterbegrenzung.

- Treffen zwei Objekte aufeinander, so wird die ***Selbst***–Maske des linken oder oberen Objektes mit der ***Stoß***–Maske des rechten oder unteren Objektes logisch UND–verknüpft. Ist das Ergebnis Null, tritt kein Unterbrechungsereignis ein, ist es von Null verschieden, tritt ein Unterbrechungsereignis ein (s.a. COLLISION– und ON COLLISION GOSUB–Anweisungen sowie Kapitel 6.4)

- Weitere Informationen zur Definition der Kollisionsmasken bei grafischen Objekten finden Sie im AmigaROM–Handbuch.

OBJECT.ON– und OBJECT.OFF–Anweisungen

Format: **OBJECT.ON** [***Objekt***[,***Objekt***. . .]]
OBJECT.OFF [***Objekt***[,***Objekt***. . .]]

Bedeutung: Diese beiden Anweisungen machen ein Objekt, mehrere oder alle Objekte im aktuellen Ausgabefenster sichtbar (OBJECT.ON) oder unsichtbar (OBJECT.OFF).

Objekt Ein ganzzahliger Ausdruck, dessen Wert größer als 0 sein muß, und der der Objekt-Kennung in der OBJECT.SHAPE-Anweisung (s. dort) entspricht. ***Objekt*** bezeichnet das Objekt im aktuellen Ausgabefenster, das sichtbar oder unsichtbar gemacht werden soll. Wird ***Objekt*** nicht angegeben, so werden alle Objekte im aktuellen Ausgabefenster sichtbar oder unsichtbar gemacht.

Beispiel: Siehe OBJECT.SHAPE–Anweisung.

Anmerkungen:

- War ein mit der OBJECT.ON–Anweisung sichtbar gemachtes Objekt vorher mit einer OBJECT.START–Anweisung (s. dort) in Bewegung gesetzt worden, so bewegt es sich nach der Sichtbarmachung weiter.
- Die OBJECT.OFF–Anweisung hält außerdem ein ggf. vorher mit OBJECT.START in Bewegung gesetztes Objekt an, wodurch auch zukünftige Kollisionen unterbunden werden.

OBJECT.PLANES–Anweisung

Format: **OBJECT.PLANES** ***Objekt*** [,***Bitebene***][,***Ebene–Ein–Aus***]

Bedeutung: Definiert zwei 8–Bit–Masken, mit denen die Farbdarstellung von Bobs (s.a. Kapitel 7.4) beeinflußt werden kann.

Objekt Ein ganzzahliger Ausdruck, dessen Wert größer als 0 sein muß, und der der Objekt-Kennung in der OBJECT.SHAPE-Anweisung (s. dort) entspricht. ***Objekt*** bezeichnet das Objekt im aktuellen Ausgabefenster, für das die Farbdarstellung geändert werden soll.

Bitebene Ein ganzahliger Ausdruck, dessen Wert zwischen 0 uns 255 liegen muß, und der das Dezimaläquivalent einer 8–Bit-Maske darstellt. Diese Maske legt fest, in welchen Bitebenen das System die Abbildung des durch ***Objekt*** bezeichneten Bobs zeichnen soll.

Ebene–Ein–Aus Ein ganzahliger Ausdruck, dessen Wert zwischen 0 und 255 liegen muß, und der das Dezimaläquivalent einer 8–Bit–Maske darstellt. Diese Maske legt fest, wie die Bildpunkte in den einzelnen Bitebenen, die nicht von der Maske ***Bitebene*** beeinflußt werden, dargestellt werden sollen.

Anmerkungen:

- Voreingestellt sind für beide Masken die Werte, die der Objekt–Editor (s. Kapitel 7) festlegt.
- Im AmigaROM–Handbuch finden Sie detaillierte Informationen zu diesen beiden Masken.

OBJECT.PRIORITY–Anweisung

Format: **OBJECT.PRIORITY** ***Objekt,Prior***

Bedeutung: Setzt die Priorität, mit der ein spezifiziertes Bob im Vergleich zu anderen Bobs mit niedrigerer oder höherer Priorität im aktuellen Ausgabefenster vom System gezeichnet wird.

Objekt Ein ganzzahliger Ausdruck, dessen Wert größer als 0 sein muß, und der der Objekt-Kennung in der OBJECT.SHAPE-Anweisung (s. dort) entspricht. ***Objekt*** bezeichnet das Bob im aktuellen Ausgabefenster, für das die Abbildungspriorität gesetzt werden soll.

Prior Ein ganzzahliger Ausdruck, dessen Wert zwischen –32768 und +32767 liegen muß, und der die Priorität, mit der das System das Abbild eines Bobs im Ausgabefenster aufbaut, festlegt. Je höher dieser Wert ist, umso höher ist die Priorität. Beispielsweise wird ein Bob mit einer Priorität von 8 vor anderen Objekten mit Prioritäten von 0 bis 7 abgebildet.

Anmerkungen:

- Bobs mit gleicher Priorität werden in zufälliger Reihenfolge abgebildet.

OBJECT.SHAPE–Anweisung

Formate:

Syntax 1: **OBJECT.SHAPE** ***Objekt,Definition***

Syntax 2: **OBJECT.SHAPE** ***Objekt 1,Objekt 2***

Bedeutung:

Syntax 1: Definiert Form, Farben, Ort und weitere Attribute für ein grafisches Objekt (Bob oder Sprite), das im aktuellen Ausgabefenster beliebig bewegt werden kann. Einzelheiten zu Bobs und Sprites finden Sie im Kapitel 7 sowie im AmigaROM–Handbuch im Abschnitt über grafische Animation.

Objekt Ein ganzzahliger Wert, der größer als 0 sein muß und der dem zu definierenden Objekt zugeordnet wird. Mit dieser Kennung nehmen die anderen OBJECT.–Anweisungen auf das jeweilige Objekt Bezug.

Definition Ein Zeichenkettenausdruck, der die statischen Attribute wie Größe, Form und Farbe des Objektes definiert. Der Objekt–Editor, den Sie in der BasicDemos–Schublade auf der Extras–Diskette finden, und der in Amiga Basic geschrieben ist, setzt diese Zeichenkette zusammen (s. Kapitel 7).

Syntax 2: Die Form von ***Objekt 2*** wird auf ***Objekt 1*** kopiert und damit ein neues Objekt erzeugt. Da beide Objekte einen beträchtlichen Speicherbreich gemeinsam belegen, kann der Speicherbedarf für multiple Objekte mit der Syntax 2 reduziert werden.

Obwohl ***Objekt 1*** und ***Objekt 2*** Speicherplatz gemeinsam belegen, können ihnen mit den anderen OBJECT–Anweisungen jeweils unterschiedliche Attribute gegeben werden. Aus diesem Grund initialisiert Amiga Basic die OBJECT.X, OBJECT.Y, OBJECT.VX, OBJECT.VY, OBJECT.AX und OBJECT.AY zugewiesenen Werte mit 0.

Beispiel:

```
OPEN "Ball" FOR INPUT AS 1
OBJECT.SHAPE 1,INPUT$(LOF(1),1)
```

Bei diesem Beispiel enthält die Datei Ball, die z.B. mit dem Objekt–Editor (s. Kapitel 7) erstellt wurde, die statischen Attribute wie Größe, Form und Farben.

Das folgende Beispiel zeigt, wie das im obigen Beispiel mit Hilfe der Datei Ball definierte Objekt bewegt werden kann, und wie Kollisionen programmiert verarbeitet werden können.

```
WINDOW 4,"Animation",(310,95)-(580,170),15
ON COLLISION GOSUB Abprallen
COLLISION ON
OPEN "Ball" FOR INPUT AS 1
OBJECT.SHAPE 1,INPUT$(LOF(1),1)
CLOSE 1
OBJECT.X 1,10
OBJECT.Y 1,50
OBJECT.VX 1,30
OBJECT.VY 1,30
OBJECT.ON
OBJECT.START
WHILE 1
    SLEEP
WEND

Abprallen:
    MerkF=WINDOW(1)
    WINDOW 4
    i=COLLISION(0)
    IF i=0 THEN RETURN
    j=COLLISION(i)
    IF j=-2 OR j=-4 THEN
        'Objekt prallt gegen linken oder rechten Rand
        OBJECT.VX i,-OBJECT.VX(i)
    ELSE
        'Objekt prallt gegen oberen oder unteren Rand
        OBJECT.VY i,-OBJECT.VY(i)
    END IF
    OBJECT.START
    WINDOW MerkF
RETURN
```

OBJECT.START– und OBJECT.STOP–Anweisungen

Formate: **OBJECT.START** [***Objekt***[,***Objekt***. . .]]
OBJECT.STOP [***Objekt***[,***Objekt***. . .]]

Bedeutung: Diese beiden Anweisungen setzen ein Objekt, mehrere oder alle Objekte im aktuellen Ausgabefenster in Bewegung (OBJECT.START) oder halten sie an (OBJECT.STOP).

Objekt Ein ganzzahliger Ausdruck, dessen Wert größer als 0 sein muß, und der der Objekt-Kennung in der OBJECT.SHAPE-Anweisung (s. dort) entspricht. ***Objekt*** bezeichnet das Objekt im aktuellen Ausgabefenster, das in Bewegung gesetzt oder angehalten werden soll. Wird ***Objekt*** nicht angegeben, so werden alle Objekte im aktuellen Ausgabefenster in Bewegung gesetzt oder angehalten.

Beispiel: Siehe OBJECT.SHAPE–Anweisung.

Anmerkungen:

- Kollidieren zwei Objekte miteinander, so führt Amiga Basic die OBJECT.STOP–Anweisung für beide Objekte automatisch aus.
- Kollidiert ein Objekt mit der Fensterbegrenzung, so führt Amiga Basic für dieses Objekt die OBJECT.STOP–Anweisung automatisch aus.

OBJECT.VX– und OBJECT.VY–Anweisungen

Formate: **OBJECT.VX** ***Objekt,Geschw***
OBJECT.VY ***Objekt,Geschw***

Bedeutung: Definiert für ein angegebenes Objekt die Geschwindigkeit in X– oder Y–Richtung.

Objekt Ein ganzzahliger Ausdruck, dessen Wert größer als 0 sein muß, und der der Objekt-Kennung in der OBJECT.SHAPE-Anweisung (s. dort) entspricht. ***Objekt*** bezeichnet das Objekt im aktuellen Ausgabefenster, für das die angegebene Geschwindigkeit gelten soll.

Geschw Ein ganzzahliger Ausdruck, dessen Wert zwischen –32768 und 32767 liegen muß, und der die Geschwindigkeit in Bildpunkten pro Sekunde definiert.

Beispiel: Siehe OBJECT.SHAPE–Anweisung.

Anmerkungen:
- Objekt können auch beschleunigt werden. Siehe dazu OBJECT.AX– und OBJECT.AY–Anweisungen.

OBJECT.VX– und OBJECT.VY–Funktionen

Formate: ***v*= OBJECT.VX (*Objekt*)**
***v*= OBJECT.VY (*Objekt*)**

Bedeutung: Liefert für ein angegebenes Objekt die Geschwindigkeit in X– oder Y–Richtung als ganzzahligen Wert in Bildpunkten/Sekunde.

Objekt Ein ganzzahliger Ausdruck, dessen Wert größer als 0 sein muß, und der der Objekt-Kennung in der OBJECT.SHAPE-Anweisung (s. dort) entspricht. ***Objekt*** bezeichnet das Objekt im aktuellen Ausgabefenster, dessen Geschwindigkeit abgefragt werden soll.

Beispiel: Siehe OBJECT.SHAPE–Anweisung.

OBJECT.X– und OBJECT.Y–Anweisungen

Formate: **OBJECT.X** ***Objekt,x***
OBJECT.Y ***Objekt,y***

Bedeutung: Plaziert ein angegebenes Objekt bei einer angegebenen Position im aktuellen Ausgabefenster (z.B. als Ausgangspunkt für Bewegungen. Die angegebenen Koordinaten bezeichnen die linke obere Ecke des Rechteckes, in dem das Objekt definiert ist.

Objekt Ein ganzzahliger Ausdruck, dessen Wert größer als 0 sein muß, und der der Objekt-Kennung in der OBJECT.SHAPE-Anweisung (s. dort) entspricht. ***Objekt*** bezeichnet das Objekt im aktuellen Ausgabefenster, das bei der angegebenen Position plaziert werden soll.

x,y Ganzzahlige Ausdrücke, deren Werte die Koordinaten der oberen linken Ecke des Objekt–Rechteckes bezeichnen.

Beispiel: Siehe OBJECT.SHAPE–Anweisung.

OBJECT.X– und OBJECT.Y–Funktionen

Formate: ***v*= OBJECT.X (*Objekt*)**
***v*= OBJECT.Y (*Objekt*)**

Bedeutung: Liefert die Koordinaten der linken oberen Ecke des Rechteckes, in dem das angegebene Objekt definiert ist, bei der aktuellen Position.

Objekt Ein ganzzahliger Ausdruck, dessen Wert größer als 0 sein muß, und der der Objekt-Kennung in der OBJECT.SHAPE-Anweisung (s. dort) entspricht. ***Objekt*** bezeichnet das Objekt im aktuellen Ausgabefenster, dessen Position abgefragt werden soll.

OCT$–Funktion

Format: ***v$*** **= OCT$(*n*)**

Bedeutung: Es wird eine Zeichenkette übergeben, die die oktale Darstellung des Wertes des numerischen Ausdruckes ***n*** enthält.

n Beliebiger numerischer Ausdruck, dessen ganzzahliger Wert gewandelt wird und der im Bereich zwischen –32768 und 65535 liegen muß. Bei negativen Werten wird die Zweier–Komplement–Darstellung benutzt.

Beispiel:

```
X=23.4:Y=-256
PRINT X;OCT$(X),Y;OCT$(Y)
```

ergibt im Ausgabefenster die Anzeige

```
23.4 27                -256 177400
Ok
```

Anmerkungen:

- Vor der Wandlung werden Dezimalstellen rechts vom Dezimalpunkt abgeschnitten.
- Die Wandlung in hexadezimale Darstellung kann mit der HEX$–Funktion (s. dort) erreicht werden.

ON BREAK–Anweisung

Format: **ON BREAK GOSUB *Marke***

Bedeutung: Das Programm verzweigt zu der angegebenen Zeile, sobald die Tastenkombinationen CTRL–C oder Amiga–. (Programmunterbrechung) gedrückt werden, oder **Stop** aus dem **Run**–Menü gewählt wird.

Marke Eine gültige Zeilennummer oder alphanumerische Sprungmarke, zu der verzweigt werden soll, sobald das Programm unterbrochen wird. Wird für Marke der Wert 0 (Null) angegeben, wird die Unterbrechungsreaktionsfähigkeit inaktiviert.

Beispiel:

```
ON BREAK GOSUB 100
BREAK ON
10 GOTO 10
    .
    .
    .
100 PRINT "Unterbrechung nicht erlaubt !"
    RETURN
```

Anmerkungen:

- Um die ON BREAK–Anweisung zu aktivieren, muß außerdem die Anweisung BREAK ON (s. dort) ausgeführt werden. Danach prüft Amiga Basic, falls die in der ON BREAK–Anweisung angegebene Marke nicht 0 war, vor der Ausführung jeder Anweisung, ob CTRL–C gedrückt oder Stop aus dem Run–Menü gewählt wurde. Ist dies des Fall, so wird zu der durch ***Marke*** spezifizierten Unterbrechungsroutine verzweigt.

- Amiga Basic führt bei einer Unterbrechungsverzweigung außerdem automatisch BREAK STOP (s.a. dort) aus, um Unterbrechungsschachtelung zu vermeiden.

- Die Ausführung der RETURN–Anweisung in der Unterbrechungsroutine bewirkt die automatische Ausführung der BREAK ON–Anweisung (s.a. dort), es sei denn, die Unterbrechungsroutine enthält die Anweisungen BREAK OFF oder BREAK STOP.

- Man kann durch Angabe einer Marke in der RETURN–Anweisung der Unterbrechungsroutine zu einer bestimmten Programmzeile des Hauptprogramms zurückverzweigen. Hier ist jedoch Vorsicht geboten, da ggf. bei der Unterbrechung aktive FOR–, GOSUB– oder WHILE–Anweisungen aktiv bleiben und dadurch der Stapelspeicher von Amiga Basic in Unordnung gebracht werden kann.

- Weitere Hinweise zur Verarbeitung von Unterbrechungsereignissen finden Sie im Kapitel 6.4.

ON COLLISION–Anweisung

Format: **ON COLLISION GOSUB *Marke***

Bedeutung: Das Programm verzweigt zu der angegebenen Zeile, sobald ein grafisches Objekt (Bob oder Sprite; s. dazu Kapitel 7.4) mit der Begrenzung des aktuellen Ausgabefensters oder einem anderen Objekt kollidiert.

Marke Eine gültige Zeilennummer oder alphanumerische Sprungmarke, zu der verzweigt werden soll, sobald eine Kollision stattgefunden hat. Wird für Marke der Wert 0 (Null) angegeben, wird die Unterbrechungsreaktion inaktiviert.

Beispiel: Siehe OBJECT.SHAPE–Anweisung.

Anmerkungen:

- Um die ON COLLISION–Anweisung zu aktivieren, muß außerdem die Anweisung COLLISION ON (s. dort) ausgeführt werden. Danach prüft Amiga Basic, falls die in der ON COLLISION–Anweisung angegebene Marke nicht 0 war, vor der Ausführung jeder Anweisung, ob eine Objektkollision stattgefunden hat. Ist dies des Fall, so wird zu der durch ***Marke*** spezifizierten Unterbrechungsroutine verzweigt.

- Amiga Basic führt bei einer Unterbrechungsverzweigung außerdem automatisch COLLISION STOP (s. dort) aus, um Unterbrechungsschachtelung zu vermeiden.

- Die Ausführung der RETURN-Anweisung in der Unterbrechungsroutine bewirkt die automatische Ausführung der COLLISION ON-Anweisung (s. dort), es sei denn, die Unterbrechungsroutine enthält die Anweisungen COLLISION OFF oder COLLISION STOP.

- Man kann durch Angabe einer Marke in der RETURN–Anweisung der Unterbrechungsroutine zu einer bestimmten Programmzeile des Hauptprogramms zurückverzweigen. Hier ist jedoch Vorsicht geboten, da ggf. bei der Unterbrechung aktive FOR–, GOSUB– oder WHILE–Anweisungen aktiv bleiben und dadurch der Stapelspeicher von Amiga Basic in Unordnung gebracht werden kann.

- Weitere Hinweise zur Verarbeitung von Unterbrechungsereignissen finden Sie im Kapitel 6.4.

ON ERROR–Anweisung

Format: **ON ERROR GOTO *Marke***

Bedeutung: Das Programm verzweigt zu einer angegebenen Zeile sobald der Amiga Basic eine Fehlermeldung absetzen will.

Marke Eine gültige Programmzeilennummer oder alphanumerische Sprungmarke, bei der die Unterbrechungsroutine beginnt. Wird für ***Marke*** 0 (Null) angegeben, so wird die Fehlerunterbrechungsreaktionsfähigkeit inaktiviert. Nachfolgende Fehler führen dann zum Programmabbruch und zur Fehleranzeige auf dem Bildschirm.

Beispiel: siehe ERR–Variable.

Anmerkungen:

- Steht in der Fehlerbehandlungsroutine die Anweisung ON ERROR GOTO 0, so wird das Programm beendet und die Fehlermeldung angezeigt. Dies erlaubt eine Programmbeendigung, falls die Fehlerursache mit Programmitteln nicht behoben werden kann.

- Da die Fehlerbehandlungsroutine nicht selbst unterbrochen werden kann, wird das Programm abgebrochen, wenn während der Fehlerbearbeitung ein weiterer Fehler auftritt.

- Existiert die in der Anweisung angegebene Zeilennummer nicht, so wird der Fehler

  ```
  Undefined line number
  ```

  (nicht definierte Zeilennummer) angezeigt.

- Um aus der Fehlerbehandlungsroutine zurück ins Hauptprogramm zu verzweigen, wird die RESUME–Anweisung (s. dort) verwendet.

ON . . . GOSUB– und ON . . . GOTO–Anweisungen

Formate: **ON *n* GOSUB *Marke*[,*Marke*]. . .**
ON *n* GOTO *Marke*[,*Marke*]. . .

Bedeutung: Das Programm verzweigt abhängig vom Wert des ganzzahligen Ausdruckes ***n*** zu der Zeile, deren Nummer oder Marke an ***n***ter Position in der Liste der spezifizierten Marken steht.

n Beliebiger ganzahliger Ausdruck, dessen Wert zwischen 0 und 255 liegen muß. Ist der Ausdruck nicht ganzzahlig, so wird sein Wert gerundet.

Marke Eine gültige Programmzeilennummer oder alphanumerische Sprungmarke, zu der verzweigt werden soll. Bei der ON . . . GOSUB–Anweisung muß dies eine Zeilennummer oder Sprungmarke einer Subroutine sein.

Beispiel 1:

```
Warte:
    Z$=INKEY$:IF Z$="" THEN Warte
    IF Z$<"A" OR Z$>"C" THEN Warte
ON ASC(Z$)-64 GOSUB Sub1,Sub2,Sub3

Sub1:
REM Beginn 1. Subroutine
.
.
RETURN

Sub2:
REM Beginn 2. Subroutine
.
.
RETURN

Sub3:
REM Beginn 3. Unterprogramm
.
.
RETURN
```

Abhängig vom Zeichencode einer der drei Tasten A, B oder C verzweigt das Programm in ein jeweils anderes Unterprogramm. Die jeweilige RETURN–Anweisung verzweigt zurück zu der auf die ON . . . GOSUB–Anweisung folgenden Anweisung.

Beispiel 2:

```
100 ON X-2 GOTO 200,300,400
110 PRINT "Keine Verzweigung"
```

Das Programm verzweigt für

X = 3 nach Zeile 200
X = 4 nach Zeile 300
X = 5 nach Zeile 400
X > 5 nach Zeile 110
X < 3 nach Zeile 110

Anmerkungen:

- Liegt *n* außerhalb des gültigen Bereiches, wird die Fehlermeldung

  ```
  Illegal function call
  ```

  (unerlaubter Funktionsaufruf) angezeigt.

ON MENU–Anweisung

Format: **ON MENU GOSUB *Marke***

Bedeutung: Das Programm verzweigt zu der angegebenen Zeile, sobald mit der Maus ein Menü aus der Menü–Leiste gewählt wird.

Marke Eine gültige Zeilennummer oder alphanumerische Sprungmarke, zu der verzweigt werden soll, sobald ein Menü gewählt wird. Wird für Marke der Wert 0 (Null) angegeben, wird die Unterbrechungsreaktionsfähigkeit inaktiviert.

Beispiel: Siehe MENU–Anweisung

Anmerkungen:

- Um die ON MENU–Anweisung zu aktivieren, muß außerdem die Anweisung MENU ON (s. dort) ausgeführt werden. Danach prüft Amiga Basic, falls die in der ON MENU–Anweisung angegebene Marke nicht 0 war, vor der Ausführung jeder Anweisung, ob ein Menü in der Menüleiste gewählt wurde. Ist dies des Fall, so wird zu der durch ***Marke*** spezifizierten Unterbrechungsroutine verzweigt.

- Amiga Basic führt bei einer Unterbrechungsverzweigung außerdem automatisch MENU STOP (s. dort) aus, um Unterbrechungsschachtelung zu vermeiden.

- Die Ausführung der RETURN–Anweisung in der Unterbrechungsroutine bewirkt die automatische Ausführung der MENU ON–Anweisung (s. dort), es sei denn, die Unterbrechungsroutine enthält die Anweisungen MENU OFF oder MENU STOP.

- Man kann durch Angabe einer Marke in der RETURN–Anweisung der Unterbrechungsroutine zu einer bestimmten Programmzeile des Hauptprogramms zurückverzweigen. Hier ist jedoch Vorsicht geboten, da ggf. bei der Unterbrechung aktive FOR–, GOSUB– oder WHILE–Anweisungen aktiv bleiben und dadurch der Stapelspeicher von Amiga Basic in Unordnung gebracht werden kann.

- Weitere Hinweise zur Verarbeitung von Unterbrechungsereignissen finden Sie im Kapitel 6.4.

ON MOUSE–Anweisung

Format: **ON MOUSE GOSUB *Marke***

Bedeutung: Das Programm verzweigt zu der angegebenen Zeile, sobald die Auswahl–Taste (linke Taste) der Maus gedrückt wird.

Marke Eine gültige Zeilennummer oder alphanumerische Sprungmarke, zu der verzweigt werden soll, sobald die Auswahltaste der Maus gedrückt wird. Wird für Marke der Wert 0 (Null) angegeben, wird die Unterbrechungsreaktionsfähigkeit inaktiviert.

Beispiel: Siehe MENU–Anweisung

Anmerkungen:

- Um die ON MOUSE–Anweisung zu aktivieren, muß außerdem die Anweisung MOUSE ON (s. dort) ausgeführt werden. Danach prüft Amiga Basic, falls die in der ON MOUSE–Anweisung angegebene Marke nicht 0 war, vor der Ausführung jeder Anweisung, ob die Auswahltaste der Maus gedrückt wurde. Ist dies des Fall, so wird zu der durch ***Marke*** spezifizierten Unterbrechungsroutine verzweigt.

- Amiga Basic führt bei einer Unterbrechungsverzweigung außerdem automatisch MOUSE STOP (s. dort) aus, um Unterbrechungsschachtelung zu vermeiden.

- Die Ausführung der RETURN–Anweisung in der Unterbrechungsroutine bewirkt die automatische Ausführung der MOUSE ON–Anweisung (s. dort), es sei denn, die Unterbrechungsroutine enthält die Anweisungen MOUSE OFF oder MOUSE STOP.

- Man kann durch Angabe einer Marke in der RETURN–Anweisung der Unterbrechungsroutine zu einer bestimmten Programmzeile des Hauptprogramms zurückverzweigen. Hier ist jedoch Vorsicht geboten, da ggf. bei der Unterbrechung aktive FOR–, GOSUB– oder WHILE–Anweisungen aktiv bleiben und dadurch der Stapelspeicher von Amiga Basic in Unordnung gebracht werden kann.

- Weitere Hinweise zur Verarbeitung von Unterbrechungsereignissen finden Sie im Kapitel 6.4.

ON TIMER–Anweisung

Format: **ON TIMER(*n*) GOSUB *Marke***

Bedeutung: Das Programm verzweigt zu der angegebenen Zeile, sobald eine angegebene Zahl von Sekunden verstrichen ist. Die Unterbrechung erfolgt alle ***n*** Sekunden.

n Ein beliebiger numerischer Ausdruck, dessen ganzzahliger Wert zwischen 1 und 86400 (24 Stunden) liegen muß, und der die Zeitspanne in Sekunden definiert, an deren Ende die Unterbrechung erfolgen soll.

Marke Eine gültige Zeilennummer oder alphanumerische Sprungmarke, zu der verzweigt werden soll, sobald die angegebene Zeit verstrichen ist. Wird für Marke der Wert 0 (Null) angegeben, wird die Unterbrechungsreaktionsfähigkeit inaktiviert.

Beispiel:

```
ON TIMER(60) GOSUB ZeigeZeit
TIMER ON
    .
    .
    .
ZeigeZeit:
    Zeile=CSRLIN:Spalte=POS(0)
    LOCATE 1,50:PRINT TIME$
    LOCATE Zeile,Spalte
RETURN
```

Alle 60 Sekunden wird die aktuelle Uhrzeit oben rechts im aktuellen Ausgabefenster angezeigt.

Anmerkungen:

- Um die ON TIMER–Anweisung zu aktivieren, muß außerdem die Anweisung TIMER ON (s. dort) ausgeführt werden. Danach prüft Amiga Basic, falls die in der ON TIMER–Anweisung angegebene Marke nicht 0 war, vor der Ausführung jeder Anweisung, ob die angegebene Zeitspanne verstrichen ist. Ist dies des Fall, so wird zu der durch ***Marke*** spezifizierten Unterbrechungsroutine verzweigt.

- Amiga Basic führt bei einer Unterbrechungsverzweigung außerdem automatisch TIMER STOP (s. dort) aus, um Unterbrechungsschachtelung zu vermeiden.

- Die Ausführung der RETURN–Anweisung in der Unterbrechungsroutine bewirkt die automatische Ausführung der TIMER ON–Anweisung (s. dort), es sei denn, die Unterbrechungsroutine enthält die Anweisungen TIMER OFF oder TIMER STOP.

- Man kann durch Angabe einer Marke in der RETURN–Anweisung der Unterbrechungsroutine zu einer bestimmten Programmzeile des Hauptprogramms zurückverzweigen. Hier ist jedoch Vorsicht geboten, da ggf. bei der Unterbrechung aktive FOR–, GOSUB– oder WHILE–Anweisungen aktiv bleiben und dadurch der Stapelspeicher von Amiga Basic in Unordnung gebracht werden kann.

- Weitere Hinweise zur Verarbeitung von Unterbrechungsereignissen finden Sie im Kapitel 6.4.

OPEN–Anweisung

Formate: **OPEN** ***Dateiang*** [**FOR** ***Modus1***] **AS** [#]***Dateinr*** [**LEN** = ***Satzlänge***]

OPEN ***Modus2***,[#]***Dateinr***,***Dateiang*** [,***Satzlänge***]

Bedeutung: Es wird eine Datei oder eine Ein–/Ausgabe–Einheit zur Dateiein– oder –ausgabe geöffnet. Beide angegebenen Formate sind gleichwertig.

Modus1 Als Parameter kann hier ein Zeichenkettenausdruck angegeben werden, dessen Wert den Datenübertragungsmodus bestimmt und einer der folgenden Ausdrücke sein muß:

OUTPUT für sequentielle Ausgabe;
APPEND für sequentielle Ausgabe, die an eine bestehende Datei angefügt wird;
INPUT für sequentielle Eingabe.

Wird **FOR** ***Modus1*** weggelassen, so wird die Datei für wahlfreien Zugriff geöffnet. Die Angaben für ***Modus1*** dürfen **nicht** in Anführungszeichen stehen.

Modus2 Als Parameter kann hier ein beliebiger Zeichenkettenausdruck angegeben werden, dessen erstes Zeichen den Datenübertragungsmodus bestimmt und eines der folgenden sein muß:

O für sequentielle Ausgabe
A für sequentielle Ausgabe, die an eine bestehende Datei angefügt wird;
I für sequentielle Eingabe;
R für Ein–/Ausgabe im wahlfreien Zugriff.

Dateinr Ein ganzzahliger Ausdruck, dessen Wert zwischen 1 und 255 liegen muß, und der den Zusammenhang zwischen Datei und dateibezogenen Ein–/Ausgabeanweisungen herstellt, solange die Datei offen ist.

Dateiang Ein Zeichenkettenausdruck für eine Dateispezifikation, wie in Kapitel 5.2 beschrieben.

Satzlänge Ein ganzzahliger Ausdruck, der, falls angegeben, zwischen 1 und 32767 liegen muß und die Länge eines logischen Datensatzes für Dateien mit wahlfreiem Zugriff spezifiziert. Wird dieser Parameter bei der OPEN–Anweisung für sequentielle Dateien angegeben, so korrespondiert er nicht notwendigerweise mit der Länge eines individuellen Satzes, da sequentielle Dateien Sätze unterschiedlicher Länge enthalten können. Hier gibt eine ggf. angegebene Satzlänge vor, wieviele Zeichen in den Dateipuffer geschrieben werden, ehe dieser auf Disk übertragen wird. Je größer also der Puffer ist, umso größer ist auch der Speicherbedarf, aber umso schneller wird auch die Datei–Ein–/Ausgabe abgewickelt. Voreingestellt ist eine Länge von 128 Bytes.

Beispiel 1:

```
OPEN "TEXT.TXT" FOR OUTPUT AS #2
```

oder

```
OPEN "O",#2,"TEXT.TXT"
```

Beide Anweisungen öffnen eine sequentielle Ausgabedatei TEXT.TXT auf dem aktuellen Diskettenlaufwerk unter der logischen Dateinummer 2.

Beispiel 2:

```
OPEN "TEXT.TXT" FOR APPEND AS #2
```

Der bestehenden sequentiellen Datei TEXT.TXT sollen Daten hinzugefügt werden.

Beispiel 3:

```
100 OPEN "DF1:INDEX" AS 1 LEN=512
```

oder

```
OPEN "R",1,"DF1:INDEX",512
```

Durch beide gleichwertigen Anweisungen wird die Datei INDEX auf der Diskette in Laufwerk DF1: für wahlfreien Zugriff mit einer logischen Satzlänge von 512 Bytes geöffnet.

Beispiel 4:

```
OPEN "DF1:VERWALTUNG/PERSONAL/GEHALT"
FOR OUTPUT AS #2
```

oder

```
PFAD$="DF1:VERWALTUNG/PERSONAL/GEHALT"
OPEN "O",#2,PFAD$
```

Es wird eine sequentielle Datei GEHALT im Diskettenunterverzeichnis PERSONAL auf der Diskette im Laufwerk DF1: geöffnet.

Anmerkungen:

- Das Öffnen einer sequentiellen Ausgabedatei zerstört alle ggf. vorher in dieser Datei abgelegten Daten. Sollen die Daten erhalten bleiben, muß die Datei im Modus APPEND eröffnet werden.

- Durch die OPEN–Anweisung wird der Datei oder Ein–/Ausgabe–Einheit ein Puffer zugeordnet und der Pufferzugriffs–Modus festgelegt.

- Folgenden Ein–/Ausgabeanweisungen, die eine Dateinummer benötigen, muß eine OPEN–Anweisung vorangehen:

INPUT #	PRINT #
LINE INPUT #	PRINT # USING
GET	PUT
	WRITE

- Disketten– oder Festplattendateien können für sequentiellen oder wahlfreien Zugriff geöffnet werden.

- Drucker können für sequentielle oder wahlfreie Ausgabe geöffnet werden. Die Satzlänge wird hier jedoch in jedem Fall auf 1 gesetzt.

- Alle anderen Ein–/Ausgabe–Einheiten können nur für sequentielle Ein–/Ausabe geöffnet werden.

- Der Modus APPEND kann nur bei sequentiellen Disk–Ausgabedateien verwendet werden.

- Eine Eingabedatei kann unter verschiedenen Dateinummern und verschiedenen Modi gleichzeitig geöffnet sein, so daß mit verschiedenen Zugriffsarten gearbeitet werden kann. In diesem Fall wird jeder Dateinummer ein eigener Puffer zugeordnet.

- Ist eine Datei schon geöffnet, kann sie nicht mehr für sequentielle Ausgabe (OUTPUT oder APPEND) geöffnet werden.

- Existiert eine Datei, die für Eingabe geöffnet werden soll, nicht, so wird die Fehlermeldung

  ```
  File not found
  ```

  (Datei nicht gefunden) angezeigt.

- Öffnen von nicht existierenden Ausgabedateien führt zur Neuanlage dieser Dateien auf der spezifizierten Disk (Diskette oder Festplatte).

- Die Angabe der OPEN–Anweisung mit ungültigen Parametern führt zu der Fehlermeldung

  ```
  Illegal function call
  ```

  (unerlaubter Funktionsaufruf).

- Weitere Einzelheiten zur Disk–Datei–Ein–und Ausgabe sind in Kapitel 5 zu finden.

OPEN "COM1:. . ."–Anweisung

Format: **OPEN "COM1:**[***Baud***][,***Parit***][,***Wortl***]
[,***Stopb***]" [**FOR** ***Modus***] **AS** [#]***Dateinr***

Bedeutung: Es wird eine Datei für Datenfernübertragung über die RS232–Schnittstelle geöffnet.

Baud Eine ganzzahlige Konstante, die den Wert 110, 150, 300, 600, 1200, 1800, 2400, 4800, 9600 oder 19200 haben muß und die die Übertragungsgeschwindigkeit in Bits pro Sekunde (Baud) angibt. Voreingestellt sind 9600 Baud beim Voreinsteller Preferences (s. Amiga–Anwenderhandbuch). Der hier gewählte Wert ersetzt den mit dem Voreinsteller Preferences gewählten.

Parit Eine Buchstabenkonstante, die die Sende– und Empfangsdatenprüfung beschreibt. Gültige Buchstaben sind:

E Sende– und Empfangsdaten werden auf gerade (even) Bit-Parität geprüft.
O Sende– und Empfangsdaten werden auf ungerade (odd) Bit–Parität geprüft.
N Es wird keine (none) Paritätsprüfung durchgeführt.

Voreingestellt ist hier der Parameter E.

Wortl Eine ganzzahlige Konstante aus dem Bereich 5, 6, 7 und 8, die angibt, aus wievielen Bits ein zu übertragendes oder empfangenes Zeichen besteht. Voreingestellt ist hier der Wert 7.

Stopb Eine ganzzahlige Konstante mit dem Wert 1 oder 2, die angibt, wieviele Stop–Bits einem zu übertragenden oder empfangenen Zeichen folgen. Voreingestellt ist hier der Wert 2 bei einer Baud–Rate von 110 und 1 bei allen anderen Baud–Raten. Wurde als Wortlänge 5 gewählt, bedeutet hier der Parameter 2, daß 1 1/2 Stop–Bits jedem Datenwort folgen.

Modus Eine der beiden Zeichenkettenkonstanten

OUTPUT für sequentielle Ausgabe,
INPUT für sequentielle Eingabe.

Fehlt die Angabe **FOR *Modus***, wird wahlfreie Ein–/Ausgabe angenommen. Dieser Modus kann nicht explizit spezifiziert werden.

Dateinr Ein ganzzahliger Ausdruck, dessen Wert eine gültige Dateinummer angibt, die in den Anweisungen für Datenfernübertragung verwendet wird.

Beispiel 1:

```
OPEN "COM1:" AS #3
```

Unter der Dateinummer 3 wird eine Datenfernübertragungsdatei über die RS232–Schnittstelle mit allen Parameter–Voreinstellungen (9600 Baud, gerade Parität, 7 Datenbits, 1 Stopbit, wahlfreie Ein–/Ausgabe) geöffnet.

Beispiel 2:

```
OPEN "COM1:4800,N,8,1" AS #3
```

Es wird eine Datenfernübertragungsdatei unter der Nummer 3 über die RS232–Schnittstelle für 4800 Baud, keine Paritätsprüfung, 8 Datenbits und 1 Stopbit eröffnet.

Anmerkungen:

- Die Pufferzuordnung erfolgt analog zu den Disk–Dateien.
- Bei 8 Datenbits muß für die Parität N angegeben werden; bei 5 Datenbits muß eine Parität definiert sein. N ist dann verboten. Bei der Übertragung von rein numerischen Daten muß die Wortlänge 8 sein.
- Aufruf der OPEN "COM. . .–Anweisung mit fehlerhaften Parametern führt zu der Fehlermeldung

  ```
  Bad file name
  ```

  (Falscher Dateiname).
- Weitere Informationen zur Datenfernübertragung sind in Kapitel 5 enthalten.

OPTION BASE–Anweisung

Format: **OPTION BASE *n***

Bedeutung: Es wird der kleinste Indexwert für Feldvariable definiert.

n Ein ganzzahliger Ausdruck, dessen Wert 0 oder 1 sein muß. Voreingestellt ist der Wert 0.

Beispiel:

```
OPTION BASE 1
```

Die Feldindizes für alle dimensionierten Felder im Programm, die normalerweise von 0 an gezählt werden, werden nach dieser Anweisung von 1 an gezählt.

Anmerkungen:

- Die Anweisung OPTION BASE muß, falls sie verwendet wird, vor jeder DIM– oder jeder feldbezogenen Anweisung stehen. Andernfalls wird der Fehler

  ```
  Duplicate definition
  ```

  (doppelte Definition) angezeigt.

- Wird für ***n*** ein anderer Wert als 0 oder 1 angegeben, so wird der Fehler

  ```
  Syntax error
  ```

  (Syntax–Fehler) angezeigt.

PAINT–Anweisung

Format: **PAINT [STEP](*x,y*) [,*Farbe*,[*Rand*]]**

Bedeutung: Malt eine geschlossene Fläche in einer wählbaren Farbe aus.

x,y Die absoluten oder relativen (mit STEP) Bildschirmkoordinaten eines beliebigen Bildpunktes innerhalb des auszumalenden, umrandeten Bereiches.

Farbe Eine Farbnummer zwischen 0 und 3 entsprechend der mit einer PALETTE–Anweisung (s. dort) gesetzten Farbkennung. In dieser Farbe wird der Bereich ausgemalt. Voreingestellt ist die Vordergrundfarbe, die mit der COLOR–Anweisung (s. dort) gewählt wurde.

Rand Eine Farbnummer zwischen 0 und 3 entsprechend der mit einer PALETTE–Anweisung (s. dort) gesetzten Farbkennung. Dieser Wert spezifiziert die Randfarbe des ausgemalten Bereiches. Wird dieser Parameter weggelassen, so wird als Randfarbe dieselbe Farbe verwendet, in der auch der Bereich ausgemalt wird.

Beispiel:

```
Radius=50:x=200:y=100
Farbe=3
CIRCLE(x,y),Radius,Farbe
PAINT(x,y),Farbe
```

Es wird ein orangefarbener (falls die Farbkennung nicht mit der PALETTE–Anweisung verändert wurde) Kreis gezeichnet und ausgemalt.

Anmerkungen:

- Das Fenster, in dem sich der auszumalende Bereich befindet, muß mit einer WINDOW–Anweisung (s. dort), bei der der ***Typ*** mit einem Wert zwischen 16 und 31 spezifiziert wurde, deklariert werden.

- Siehe auch PATTERN–, AREA– und AREAFILL–Anweisungen.

PALETTE–Anweisung

Format: **PALETTE *Farbe,Rot,Grün,Blau***

Bedeutung: Definiert für eine Farbkennung, die von anderen grafischen Amiga Basic–Anweisungen verwendet wird, einen Farbton.

Farbe Ein numerischer Ausdruck, dessen ganzzahliger Wert zwischen 0 und 31 liegen muß, und der die Farbkennung für eine Farbe, die von den Anweisungen CIRCLE, COLOR, LINE, PAINT, PRESET, und PSET benutzt wird, definiert. Der Parameter ***Tiefe*** bei der SCREEN–Anweisung (s. dort) legt fest, welcher maximale Wert für ***Farbe*** angegeben werden kann. Voreingestellt ist hier der Wert 3, da das Amiga System ebenfalls voreinstellungsmäßig mit 4 Farben arbeitet, nämlich mit

0 blau
1 weiß
2 schwarz
3 orange

Jeder andere Farbton, der mit der PALETTE–Anweisung einer dieser 4 Kennungen zugewiesen wird, überschreibt den voreingestellten Farbton. In den oben genannten Amiga Basic–Anweisungen kann auf diese vier Farben Bezug genommen werden. Mit Hilfe des Voreinstellerprogramms Preferences des Arbeitstisches können diese Farben ebenfalls geändert werden.

Rot,Grün,Blau Numerische Ausdrücke, deren Werte zwischen 0.00 und 1.00 liegen müssen, und die den prozentualen Anteil der Grundfarben rot, grün und blau an der mit ***Farbe*** spezifizierten Systemfarbe definieren. Als Anhalt kann folgende Tabelle dienen, in der für eine Reihe von Farben die Rot–, Grün– und Blau–Bestandteile zusammengestellt sind:

Farbe	rot	grün	blau
blaugrün	0.00	0.93	0.87
schwarz	0.00	0.00	0.00
dunkelblau	0.40	0.60	1.00
himmelblau	0.47	0.87	1.00
braun	0.80	0.60	0.53
grau	0.73	0.73	0.73
grün	0.33	0.87	0.00
hellgrün	0.73	1.00	0.00
orange	1.00	0.73	0.00
purpur	0.80	0.00	0.93
kirschrot	1.00	0.60	0.67
feuerwehrrot	0.93	0.20	0.00
oker	1.00	0.87	0.73
violett	1.00	0.13	0.93
weiß	1.00	1.00	1.00
gelb	1.00	1.00	0.13

Beispiel:

```
Farbe:
    PALETTE 1,RND,RND,RND
    PALETTE 2,RND,RND,RND
    COLOR 1,2
    FOR i=1 TO 1000:NEXT
    GOTO Farbe
```

In einer Endlosschleife werden für Vorder– und Hintergrund zufällige Farben erzeugt und festgelegt.

PATTERN–Anweisung

Format: **PATTERN** [*Lmuster*][,*Fmuster*]

Bedeutung: Definiert das Muster, mit dem Linien und/oder Flächen gezeichnet bzw. ausgemalt werden.

Lmuster Ein numerischer Ausdruck, dessen ganzzahliger Wert das Äquivalent einer 16–Bit–Maske darstellt und der das Muster festlegt, in dem Linien gezeichnet werden (z.B. mit der CIRCLE– oder der LINE–Anweisung). Jedes gesetzte Bit (1) in dieser Maske bedeutet, daß bei einer Linie ein Bildpunkt in der aktuellen Vordergrundfarbe gezeichnet wird. Jedes gelöschte Bit (0) bedeutet, daß dieser Punkt in der Hintergrundfarbe gezeichnet wird, also nicht sichtbar ist. Voreingestellt ist hier der Wert 65535 oder &HFFFF. Linien werden also durchgezogen dargestellt.

Fmuster Ein Ganzzahlfeld, das das Muster enthält. Der Wert jedes Elementes stellt wieder das Äquivalent einer 16–Bit–Maske dar, in der 1 einen gezeichneten und 0 einen gelöschten Bildpunkt darstellt. Die Anzahl der Elemente in diesem Feld bestimmt die Höhe des Musters und muß eine Potenz von zwei sein, also 1, 2, 4 ,8, 16 usw.

Beispiel:

```
DIM Feld%(3)
Feld%(1)=&H5555
Feld%(1)=&HAAAA
Feld%(2)=&H5555
Feld%(3)=&HAAAA
PATTERN &HFFFF,Feld%
```

Bei diesem Beispiel wird für Linien das Muster für durchgezogene Striche und für Flächen ein Punktmuster definiert, das 4 Bildpunkte hoch ist.

Anmerkungen:

- Im Abschnitt "Grafische Unterstützung" des AmigaROM-Handbuches ist die Definition von Mustern ausführlich beschrieben.
- Siehe auch AREAFILL–Anweisung.

PEEK–Funktion

Format: ***v*= PEEK(*Adresse*)**

Bedeutung: Es wird der binäre Inhalt einer spezifizierten Speicherzelle als Dezimaläquivalent im Bereich zwischen 0 und 255 übergeben.

Adresse Ein ganzzahliger Ausdruck, dessen Wert im Bereich zwischen 0 und 16777215 liegen muß und der die Adresse der auszulesenden Speicherzelle spezifiziert.

Beispiel:

```
FOR i=1024 TO 2048
    IF PEEK(i)<>0 THEN PRINT PEEK(i)
NEXT
```

Die Dezimaläquivalente der Speicherinhalte im Adressbereich zwischen 1024 und 2048 werden angezeigt, sofern sie von Null verschiedenen sind.

Anmerkungen: Siehe auch PEEKL– und PEEKW–Funktionen sowie POKE–Anweisung.

PEEKL–Funktion

Format: ***v*= PEEKL(*Adresse*)**

Bedeutung: Es wird der binäre Inhalt einer spezifizierten Speicherzelle als Dezimaläquivalent im Bereich zwischen –2147483648 und 2147483647 als lange Ganzzahl übergeben.

Adresse Ein ganzzahliger Ausdruck, dessen Wert im Bereich zwischen 0 und 16777212 liegen muß und der die Adresse der auszulesenden Speicherzeile spezifiziert. ***Adresse*** muß eine gerade Zahl sein, andernfalls wird die Fehlermeldung

```
Illegal function call
```

(Unerlaubter Funktionsaufruf) angezeigt.

Anmerkungen:

- Da das Ergebnis als lange Ganzzahl interpretiert wird, können auch negative Werte übergeben werden. In diesen Fällen ist das oberste Bit gesetzt.

PEEKW–Funktion

Format: ***v*=PEEKW(*Adresse*)**

Bedeutung: Es wird der binäre Inhalt einer spezifizierten Speicherzelle als Dezimaläquivalent im Bereich zwischen –32768 und 32767 als ganze Zahl übergeben.

Adresse Ein ganzzahliger Ausdruck, dessen Wert im Bereich zwischen 0 und 16777214 liegen muß und der die Adresse der auszulesenden Speicherzeile spezifiziert. ***Adresse*** muß eine gerade Zahl sein, andernfalls wird die Fehlermeldung

```
Illegal function call
```

(Unerlaubter Funktionsaufruf) angezeigt.

Anmerkungen:

- Da das Ergebnis als ganze Zahl interpretiert wird, können auch negative Werte übergeben werden. In diesen Fällen ist das oberste Bit gesetzt.

POINT–Funktion

Format: ***v*= POINT(*x,y*)**

Bedeutung: Liefert die Farbkennung des Bildpunktes bei den Koordinaten ***x,y*** im aktuellen Ausgabefenster. Der übergebene Wert entspricht der Farbkennung in der PALETTE–Anweisung (s. dort).

x,y Gültige Koordinaten im aktuellen Ausgabefenster.

Anmerkungen:

- Die Koordinaten (0,0) bezeichnen die linke obere Ecke des aktuellen Ausgabefensters.
- Liegen die Koordinaten ausßerhalb des gültigen Bereiches des aktuellen Ausgabefensters, so liefert POINT den Wert –1.

POKE–Anweisung

Format: **POKE *Adresse,m***

Bedeutung: Es wird der Wert ***m*** als 8–Bit–Binärwert in eine spezifizierte Speicherzelle geschrieben.

Adresse Ein ganzzahliger Ausdruck, dessen Wert im Bereich zwischen 0 und 16777215 liegen muß und die Adresse der zu beschreibenden Speicherzelle darstellt.

m Ein ganzzahliger Ausdruck, dessen Wert zwisc hen 0 und 255 liegen muß und der das Äquvalent der zu speichernden Information darstellt.

Beispiel:

```
FOR i=0 TO 511
   POKE 650000+i,65
NEXT
```

Ab Adresse 650000 werden 512 Bytes mit dem Wert 65 beschrieben.

Anmerkungen:

- Da Amiga Basic keine Adreßprüfung durchführt, muß darauf geachtet werden, daß das System nicht durch unerlaubtes Überspeichern von Arbeitsspeicherzellen (Stapelspeicher, Variablenbereich usw.) lahmgelegt wird. In einem solchen Fall muß der Amiga durch gleichzeitiges Drücken der Taste CTRL sowie der beiden Amiga–Tasten rückgesetzt, also neu gestartet werden. Ggf. speicherresidente Programme werden dadurch gelöscht.

- Siehe auch POKEL– und POKEW–Anweisungen sowie PEEK– und VARPTR–Funktionen.

POKEL–Anweisung

Format: **POKEL *Adresse,m***

Bedeutung: Es wird der Wert ***m*** als 32–Bit–Binärwert in eine spezifizierte Speicherzelle geschrieben.

Adresse Ein ganzzahliger Ausdruck, dessen Wert im Bereich zwischen 0 und 16777212 liegen muß und die Adresse der zu beschreibenden Speicherzelle darstellt. ***Adresse*** muß eine gerade Zahl sein, andernfalls wird die Fehlermeldung

```
Illegal function call
```

(unerlaubter Funktionsaufruf) angezeigt.

m Ein ganzzahliger Ausdruck, dessen Wert zwischen –2147483648 und 2147483647 liegen muß und der das Äquvalent der zu speichernden Information darstellt.

Anmerkungen:

- Da Amiga Basic keine Adreßprüfung durchführt, muß darauf geachtet werden, daß das System nicht durch unerlaubtes Überspeichern von Arbeitsspeicherzellen (Stapelspeicher, Variablenbereich usw.) lahmgelegt wird. In einem solchen Fall muß der Amiga durch gleichzeitiges Drücken der Taste CTRL sowie der beiden Amiga–Tasten rückgesetzt, also neu gestartet werden. Ggf. speicherresidente Programme werden dadurch gelöscht.

POKEW–Anweisung

Format: **POKEW *Adresse,m***

Bedeutung: Es wird der Wert ***m*** als 16–Bit–Binärwert in eine spezifizierte Speicherzelle geschrieben.

Adresse Ein ganzzahliger Ausdruck, dessen Wert im Bereich zwischen 0 und 16777214 liegen muß und die Adresse der zu beschreibenden Speicherzelle darstellt. ***Adresse*** muß eine gerade Zahl sein, andernfalls wird die Fehlermeldung

```
Illegal function call
```

(unerlaubter Funktionsaufruf) angezeigt.

m Ein ganzzahliger Ausdruck, dessen Wert zwischen –32768 und 32767 liegen muß und der das Äquvalent der zu speichernden Information darstellt.

Anmerkungen:

- Da Amiga Basic keine Adreßprüfung durchführt, muß darauf geachtet werden, daß das System nicht durch unerlaubtes Überspeichern von Arbeitsspeicherzellen (Stapelspeicher, Variablenbereich usw.) lahmgelegt wird. In einem solchen Fall muß der Amiga durch gleichzeitiges Drücken der Taste CTRL sowie der beiden Amiga–Tasten rückgesetzt, also neu gestartet werden. Ggf. speicherresidente Programme werden dadurch gelöscht.

POS–Funktion

Format: ***v*= POS(*n*)**

Bedeutung: Liefert die ungefähre Spaltenposition des Textcursors in der aktuellen Zeile des aktuellen Ausgabefensters.

Beispiel:

```
Y=CSRLIN 'merkt sich vertikale Position
X=POS(0) 'merkt sich horizontale Position
LOCATE 20,1
PRINT "Dies ist Zeile 20 (Schirm-Ende)"
LOCATE Y,X
PRINT "Dies war die ursprüngliche Cursor-Position"
```

Bei diesem Beispiel wird die alte Cursor–Position gespeichert, dann am unteren Schirmende eine Meldung angezeigt und schließlich eine weitere Meldung bei der ursprünglichen Cursor–Position angezeigt.

Anmerkungen:

- Zur Bestimmung der exakten Spaltennummer benutzt Amiga Basic die Höhe und Breite der Ziffer Null (0) des für das aktuelle Ausgabefenster gewählten Zeichensatzes.
- Der übergebene Wert ist immer größer oder gleich 1.

PRESET– und PSET–Anweisungen

Formate: **PRESET** [**STEP**] (*x,y*)[,***Farbe***]
PSET [**STEP**] (*x,y*)[,***Farbe***]

Bedeutung: Es wird an einer definierten Bildschirmposition ein Punkt gezeichnet.

x,y Die absoluten oder relativen (mit STEP) Koordinaten des zu zeichnenden Punktes.

Farbe Eine Farbnummer zwischen 0 und 3 entsprechend der mit der PALETTE–Anweisung (s. dort) gesetzten Farbkennung. Voreingestellt ist die Vordergrundfarbe, die mit der COLOR–Anweisung (s. dort) eingestellt wurde.

Der Unterschied zwischen PRESET und PSET ist der, daß, falls für PRESET kein Farb–Parameter angegeben ist, die Hintergrundfarbe ausgewählt wird.

Beispiel:

```
FOR I=0 TO 100
    PSET(I,I)
NEXT
FOR I=100 TO 0 STEP -1
    PRESET(I,I)
NEXT
```

Zunächst wird eine Bildschirmdiagonale in der voreingestellten Vordergrundfarbe gezeichnet, die daraufhin wieder gelöscht wird, indem sie punktweise auf die Hintergrundfarbe gesetzt wird.

Anmerkungen:

- Werden relative Koordinaten mit STEP angegeben, so ergibt sich die endgültige Koordinate aus der Addition der angegebenen *x*– und *y*–Werte zu der aktuellen Position des grafischen Cursors.

- Koordinatenangaben außerhalb des definierten Bildschirmbereiches werden ignoriert.

- Ungültige Farbcode–Angaben haben die Fehlermeldung

  ```
  Illegal function call
  ```

  (unerlaubter Funktionsaufruf) zur Folge.

PRINT–Anweisung

Formate: **PRINT** [***Liste von Ausdr***]

oder

?[***Liste von Ausdr***]

Bedeutung: Die Werte der Liste von Ausdrücken werden im aktuellen Ausgabefenster angezeigt.

Liste von Ausdr Eine beliebige Liste von numerischen und/oder Zeichenkettenausdrücken, deren Werte angezeigt werden sollen. Die einzelnen Ausdrücke müssen durch Komma (,), Semikolon (;), Sonderzeichen wie ≠ $ & % " oder Leerstellen voneinander getrennt sein. Die Liste muß zusammen mit dem Schlüsselwort PRINT oder der Abkürzung ? in einer Programmzeile stehen. Jede Zeichenkettenkonstante muß in Anführungszeichen (") eingeschlossen sein. Wird keine Liste von Ausdrükken angegeben, so wird eine Leerzeile auf dem Bildschirm erzeugt.

Beispiel 1:

```
A=2:B=3
PRINT A+B,A*B,A↑B

ergibt

5               6               8
Ok
```

Die Werte der drei in der PRINT–Anweisung angegebenen Ausdrücke werden jeweils am Anfang einer Druckzone von 15 Zeichen Länge (Voreinstellung) angezeigt, da die Ausdrücke in der Liste durch Komma getrennt sind (s. Anmerkungen).

Beispiel 2:

```
Eingabe:
    INPUT "Zwei Zahlen (0<Z<10)";A,B
    IF A<1 OR A>9 OR B<1 OR B>10 THEN Eingabe
    PRINT "Die Summe ist"A+B;
    PRINT " und das Produkt"A*B
```

ergibt die Anzeige:

```
Zwei Zahlen (0<Z<10) ? 5,7.2
Die Summe ist 12.2 und das Produkt 36
Ok
```

Das Semikolon bei der ersten PRINT–Anweisung bewirkt, daß beide Ergebnisse in einer Zeile angezeigt werden (s. Anmerkungen).

Beispiel 3:

```
FOR I=1 TO 4
    X=X+1:Y=Y+X↑2
    ?X;Y;
NEXT
```

ergibt die Anzeige

```
1  1  2  5  3  14  4  30
```

Die Semikolons in der PRINT–Anweisung (? ist als Abkürzung für PRINT erlaubt) bewirken, daß alle Werte hintereinander gedruckt werden. Jeder positiven Zahl wird eine Leerstelle vorangestellt, und jeder Zahl wird eine Leerstelle angefügt (s. Anmerkungen).

Anmerkungen:

- Die Position für jeden angezeigten Wert innerhalb der Anzeigezeile wird durch die Zeichen bestimmt, mit denen die einzelnen Ausdrücke in der Liste voneinander getrennt sind. Die Bildschirmzeile wird voreinstellungsmäßig in Druckzonen zu je 15 Spalten unterteilt. Die Breite der Druckzonen kann mit der WIDTH–Anweisung (s. dort) verändert werden. Ein Komma zwischen zwei Ausdrücken bewirkt, daß der Wert des dem Komma folgenden Ausdrucks auf dem Bildschirm am Anfang der nächsten Druckzone angezeigt wird (s. Beispiel 1). Eine oder mehrere Leerstellen oder ein Semikolon zwischen zwei Ausdrücken bedeutet, daß der Wert des nachfolgenden Ausdrucks unmittelbar hinter dem des vorhergehenden angezeigt wird.

- Endet die Liste von Ausdrücken mit einem Komma, Semikolon, der Funktion SPC oder TAB (s. dort), so werden die nächsten PRINT–Anweisungen von der dadurch definierten Spaltenposition an in derselben Zeile ausgeführt (s. Beispiel 2).

- Endet die Liste von Ausdrücken ohne Komma, Semikolon, SPC– oder TAB–Funktion, so wird ein Wagenrücklauf und Zeilenvorschub ausgeführt, der Cursor also an den Anfang der nächsten Zeile gesetzt.

- Ist ein anzuzeigender Wert länger als der noch verfügbare Platz in der Zeile, so wird der gesamte Wert am Anfang der nächsten Zeile angezeigt.

- Ist der Wert länger als die mit der WIDTH–Anweisung definierte Bildschirmbreite, so wird der nicht mehr in eine Zeile passende Rest am Anfang der folgenden Bildschirmzeile angezeigt.

- Wurde eine Ausgabe in der letzten Zeile des aktuellen Ausgabefensters veranlaßt, so führt ein Wagenrücklauf/ Zeilenvorschub zum Aufrollen des definierten Bildschirmfensters um eine Zeile.

- Nach angezeigten numerischen Werten wird immer eine Leerstelle angefügt. Ebenso vor positiven numerischen Werten (s. Beispiel 3); vor negativen wird ein Minuszeichen (–) eingefügt .

- Zahlen einfacher Genauigkeit, die mit sieben oder weniger Stellen im Festpunktformat dargestellt werden können, ohne an Genauigkeit zu verlieren, werden im Festpunkt– oder ganzzahligen Format angezeigt; alle anderen in der Exponentialdarstellung des Gleitpunktformats:

  PRINT 10↑–7 ergibt **.0000001**
  PRINT 10↑–8 ergibt **1E–08**

PRINT USING–Anweisung

Format: **PRINT USING** ***v$;Liste von Ausdr***[;]

Bedeutung: Die Werte der Liste von Ausdrücken werden entsprechend dem in der Anweisung angegebenen Format auf dem Bildschirm angezeigt.

v$ Eine Zeichenkettenvariable oder –konstante, die das Druckformat definiert.

Liste von Ausdr Eine beliebige Liste von numerischen und/oder Zeichenkettenausdrücken, deren Werte auf dem Bildschirm angezeigt werden sollen. Die einzelenen Ausdrücke müssen duch Komma (,), Semikolon (;) oder eine oder mehrere Leerstellen voneinander getrennt sein. Hier führt das Komma nach einem Ausdruck jedoch **nicht** zur Tabulierung auf den Anfang der nächsten Druckzone. Die Formatierung der Ausgabe erfolgt durch Steuerzeichen, deren Wirkung im folgenden getrennt für Zeichenketten– und numerische Druckfelder beschrieben wird.

Zeichenkettenfelder:

! Nur das erste Zeichen einer gegebenen Zeichenkette soll angezeigt werden.

***n* Leerstellen** Es sollen 2 + *n* Zeichen einer gegebenen Zeichenkette angezeigt werden. Ist die Zeichenkette länger als *n* Zeichen, werden die überzähligen Zeichen ignoriert. Ist sie kürzer, werden die Zeichen linksbündig angezeigt. Rechts werden Leerstellen angefügt.

Beispiel 1:

```
A$="REGEN":B$="BOGEN"
PRINT USING "!";A$;B$
PRINT USING "\  \";A$;B$
PRINT USING "\   \";A$;B$;"!!"
```

ergibt die Anzeige

```
RB
REGENBOGEN
REGEN BOGEN !!
Ok
```

& Es wird ein Druckfeld variabler Länge definiert, d.h., der anzuzeigende Zeichenkettenausdruck wird in seiner vollen Länge ausgegeben.

Beispiel 2:

```
A$="REGEN":B$="BOGEN"
PRINT USING "!";A$;
PRINT USING "&";B$
```

ergibt die Anzeige

```
RBOGEN
Ok
```

Numerische Felder

Das Nummernzeichen bezeichnet eine Ziffernposition. Hat die anzuzeigende Zahl weniger Ziffern, als dafür Positionen definiert sind, wird sie rechtsbündig mit führenden Leerstellen im Feld angezeigt. Ein Punkt (.) in der Formatzeichenkette kennzeichnet die Position des Dezimalpunktes. Soll links vom Punkt eine Ziffer ausgegeben werden, so wird, falls keine Stelle links vom Punkt angezeigt würde, eine Null (0) ausgegeben. Wird für eine gebrochene Zahl ein Formatfeld ohne Punkt angegeben, so wird die Zahl vor der Anzeige entsprechend gerundet.

Beispiel 3:

```
PRINT USING "#.##";.15
0.15
Ok

PRINT USING "##.##";12.123
12.12
Ok

PRINT USING "###.##";15.1,1.829,.321
 15.10 1.83 0.32
Ok
```

+ Ein Plus–Zeichen am Anfang oder Ende der Formatzeichenkette bewirkt je nach Wert des Ausdruckes die Ausgabe eines Plus– oder Minus–Zeichens vor oder hinter dem Wert.

– Ein Minus–Zeichen am Ende der Formatzeichenkette bewirkt die Ausgabe eines Minuszeichens hinter negativen Werten.

Beispiel 4:

```
PRINT USING "+##.##";-80.5,2.89,-.4
-80.50 +2.89 -0.40
Ok

PRINT USING "##.##-";-80.5,2.89,-.4
80.5- 2.89 0.40-
Ok
```

** Führende Leerstellen bei einem Wert werden mit Sternen aufgefüllt. Dabei stehen die beiden Sterne für Ziffernpositionen.

Beispiel 5:

```
PRINT USING "**#.##";10.46,-.9,150.3
*10.46*-0.90150.30
Ok
```

$$ Vor dem Zahlenwert wird ein Dollarzeichen angezeigt. Beide Dollarzeichen stehen für Zeichenpositionen; eines davon für das Dollarzeichen selbst. Bei der Exponentialdarstellung können die Dollarzeichen nicht verwendet werden. Ebenso bei negativen Werten, es sei denn, das Minuszeichen steht hinter der Zahl.

Beispiel 6:

```
PRINT USING "$$###.##";390.85
 $390.85
Ok
```

**$ Die beiden zuletzt beschriebenen Formatsteuerungen werden kombiniert. Führende Leerstellen werden mit Sternen aufgefüllt und es wird vor dem Wert ein Dollarzeichen ausgegeben. Diese Formatsteuerung steht selbst für drei Zeichenpositionen; eine davon ist das Dollarzeichen selbst.

Beispiel 7:

```
PRINT USING "**$##.##";1.85
***$1.85
Ok
```

, Ein Komma unmittelbar links vom Dezimalpunkt in der Formatkette bewirkt, daß in allen Tausenderpositionen links vom Dezimalpunkt ein Komma in den anzuzeigenden Wert eingefügt wird. Ein Komma am Ende der Formatkette wird als Komma an dieser Position mit ausgegeben. Das Komma definiert selbst eine Zeichenposition. Beim Exponentialformat (s. unten) hat die Angabe eines Kommas keine Wirkung.

Beispiel 8:

```
PRINT USING "####,.##";5876.7
5,876.70
Ok

PRINT USING "####.##,";5876.7
5876.70,
Ok
```

↑↑↑↑ Werden vier Potenzierungssymbole an die Formatzeichenkette angefügt, so wird dadurch das Exponentialformat (**E+*nn*** oder **E–*nn***) definiert. Es kann jede Dezimalpunktposition angegeben werden. Signifikante Ziffern werden linksbündig ausgegeben. Ist kein führendes Plus oder nachfolgendes + bzw. – angegeben, wird eine Position links vom Dezimalpunkt für das Vorzeichen vorgesehen.

Beispiel 9:

```
PRINT USING "##.##↑↑↑↑";186.27
 1.86E+02
Ok

PRINT USING ".###↑↑↑↑-";-88888
.889E+05-
Ok

PRINT USING "+.##↑↑↑↑";154
+.15E+0.3
Ok
```

_ Ein Unterstreichungsstrich bewirkt, daß das nächste Zeichen der Formatzeichenkette als Zeichen selbst ausgegeben wird.

Beispiel 10:

```
PRINT USING "_/##.##_/";13.769
/13.77/
Ok
```

Wenn die darzustellende Zahl mehr Stellen enthält, als die dafür definierte Formatzeichenkette, so wird die Zahl mit vorangestelltem Prozentzeichen (%) ausgegeben.

Beispiel 11:

```
PRINT USING "##.##";238.15
%238.15
Ok
```

Schließlich darf die Formatzeichenkette auch Zeichenkettenkonstanten enthalten:

Beispiel 12:

```
PRINT USING "Dies ist Beispiel _###";12
Dies ist Beispiel #12
Ok
```

Anmerkungen:

- Werden in der Formatzeichenkette mehr als 24 Ziffernpositionen definiert, so wird die Fehlermeldung

  ```
  Illegal function call
  ```

  (unerlaubter Funktionsaufruf) angezeigt

PRINT #– und PRINT #,USING–Anweisungen

Format: **PRINT #*Dateinr*, [USING *v$*;]*Liste von Ausdr***

Bedeutung: Es werden Daten in eine sequentielle Disk-Datei oder an eine sequentielle Einheit ausgegeben.

Dateinr Ein ganzzahliger Ausdruck, unter dessen Wert die Datei oder Einheit für sequentielle Ausgabe eröffnet wurde.

v$ Eine Zeichenkettenvariable oder –konstante, die das Ausgabeformat definiert (s. dazu PRINT USING–Anweisung).

Liste von Ausdr Eine beliebige Liste von numerischen und/oder Zeichenkettenausdrücken, deren Werte in die Datei oder an die Einheit ausgegeben werden sollen. Die einzelnen Ausdrücke müssen durch Komma (,) oder (bei numerischen Ausdrücken immer) durch Semikolon (;) getrennt werden. Da durch die PRINT #–Anweisung die Daten in der Datei nicht komprimiert werden, sondern vielmehr so ausgegeben werden, wie sie auch auf dem Bildschirm mit PRINT angezeigt werden, wird im folgenden an verschiedenen Beispielen gezeigt, wie die einzelnen Ausdrücke richtig durch Trennzeichen getrennt werden müssen, damit sie später wieder korrekt mit der INPUT #–Anweisung (s. dort) gelesen werden können.

Beispiel 1:

```
CR$=CHR$(13):A=1:B=2.5:C=4:D=28
PRINT #1,A;CR$B;CR$C;CR$D
```

Die numerischen Variablen A bis D werden mit Wagenrücklaufcodes getrennt ausgegeben, so daß sie mit INPUT #*n*,A,B,C,D wieder gelesen werden könnten. Zwischen der Variablen CR$ und der jeweils folgenden numerischen Variablen braucht kein Semikolon angegeben zu werden, da das Dollarzeichen dem Interpreter das Ende einer Zeichenkettenvariablen anzeigt. Kommas (,) als Trennzeichen führen bei PRINT # zu einer identischen Formatierung in 15 Stellen (voreingestellt, kann mit der WIDTH–Anweisung geändert werden) lange Druckfelder wie bei der Bildschirmausgabe mit PRINT (s. dort).

Beispiel 2:

```
A$="SONNENBLUMEN":B$="KERN"
PRINT #2,A$;B$
```

Hier würde SONNENBLUMENKERN ausgegeben werden, da das Semikolon nach A$ die Ausgabe eines Trennzeichens unterdrückt.

Beispiel 3:

```
A$="SONNENBLUMEN":B$="KERN":C$=","
PRINT #2,A$C$
```

In diesem Fall wird SONNENBLUMEN,KERN ausgegeben, was wieder in getrennte Zeichenkettenvariablen eingelesen werden könnte, da das Komma für INPUT # ein Trennzeichen darstellt.

Beispiel 4:

```
A$="AUTO,KFZ";:B$=" NUMMER":C$=CHR$(34)
PRINT #2,C$;A$;C$;B$;C$
```

Zeichenketten, die signifikante führende Leerstellen, Semikolons, Kommas, Wagenrücklauf– und/oder Zeilenvorschubcodes enthalten, müssen bei der Ausgabe in Anführungsstriche (" = CHR$(34)) eingekleidet werden, damit sie mit INPUT # wieder richtig gelesen werden können.

Beispiel 5:

```
100 PRINT #1,USING "_/##.##";A
```

Auch bei der Dateiausgabe können numerische Werte mit Hilfe der PRINT # USING–Anweisung formatiert ausgegeben werden.

Anmerkungen:

- Da bei der PRINT #–Anweisung die Ausgabe korrekter Trennzeichen vom Anwender genau beachtet werden muß, empfiehlt sich die sequentielle Datenausgabe mit der WRITE #–Anweisung (s. dort).

- Weitere Einzelheiten siehe Kapitel 5.

PSET–Anweisung

Siehe bei **PRESET**–Anweisung.

PTAB–Funktion

Format: **PRINT PTAB(*n*)**

Bedeutung: Die Druckposition wird in der aktuellen Zeile des aktuellen Ausgabefensters auf ***n*** Bildpunkte vom linken Rand (Bildpunkt 0) gesetzt. Befindet sich die aktuelle Druckposition rechts von Bildpunkt ***n***, so wird auf Bildpunkt ***n*** in derselben Zeile positioniert.

n Ein numerischer Ausdruck, dessen ganzzahliger Wert zwischen 0 und 32767 liegen muß.

Anmerkungen:

- Die PTAB–Funktion entspricht der TAB–Funktion (s. dort). Es wird jedoch nicht um Druckspalten, sondern um Bildpunkte tabuliert.
- Die PTAB–Funktion darf nur in Verbindung mit der PRINT-Anweisung verwendet werden.
- Steht die PTAB–Funktion am Ende einer Liste von Ausdrükken, so wird **kein** Wagenrücklauf/Zeilenvorschub ausgeführt.

PUT–Anweisung für Dateien

Format: **PUT** [#]***Dateinr***[,***Satznr***]

Bedeutung: Der Inhalt des Puffers für wahlfreien Zugriff wird als logisch nächster oder als Satz mit angegebener Satznummer in eine Datei für wahlfreien Zugriff geschrieben.

Dateinr Die Nummer, unter der die Datei für wahlfreien Zugriff in der OPEN–Anweisung (s. dort) eröffnet wurde.

Satznr Die Nummer des zu schreibenden logischen Datensatzes, die zwischen 1 und 16777215 liegen muß, wenn sie angegeben wird.

Beispiel: Siehe Beispiel bei MKI$–, MKL$–, MKS$– und MKD$–Funktionen.

Anmerkungen:

- Die Anweisungen PRINT #, PRINT # USING, WRITE #, RSET und LSET können dazu benutzt werden, Daten in den Puffer zu übertragen, ehe PUT ausgeführt wird. Bei WRITE # wird der Puffer nach den Daten bis zur definierten Satzlänge mit Leerstellen aufgefüllt.

- Der Versuch, mehr Daten in den Puffer zu übertragen, als dessen in der OPEN–Anweisung definierte Länge zuläßt, führt zu der Fehlermeldung

  ```
  Field overflow
  ```

  (Feldüberlauf).

- Das Betriebssystem AmigaDOS legt auf Diskette oder Festplatte physikalische Sätze (Sektoren) von 512 Bytes Länge an, die fortlaufend die logischen Sätze enthalten, so daß bei kürzeren logischen Sätzen mehrere in einen physikalischen Satz passen. Deshalb führt eine PUT–Anweisung nicht unbedingt zu einem physikalischen Schreibvorgang.

- Weitere Hinweise zur Dateiausgabe mit PUT werden im Kapitel 5 gegeben.

PUT–Anweisung für Grafik

Format: **PUT [STEP] (*x,y*),*Feldvar* [(*Index*[,*Index*. . .,*Index*])][,*Modus*]**

Bedeutung: Es werden Daten aus einem numerischen Feld als binäre Information entnommen und als farbige Punkte im aktuellen Ausgabefenster dargestellt.

x,y Die absoluten oder relativen (mit STEP) Koordinaten der linken oberen Ecke des aktuellen Bildschirmfensters.

Feldvar Eine numerische Feldvariable beliebiger Genauigkeit, in der die Punktinformation bitweise abgelegt ist (s. dazu GET-Anweisung für Grafik).

Index Die optionale Angabe von Feldelementen über deren Index erlaubt den Zugriff auf mehrere in einem mehrdimensionalen Feld gespeicherte grafische Objekte.

Modus Der Modus beschreibt die Art, wie die Daten auf den Bildschirm gebracht werden sollen oder wie bereits auf dem Bildschirm befindliche Daten manipuliert werden können. Es gibt folgende Modi:

PSET Die Daten aus dem Feld werden direkt auf den Bildschirm übertragen. Jeder Punkt hat dieselbe Farbe, mit der er mit der GET–Anweisung für Grafik gespeichert wurde.

PRESET Genau wie bei PSET. Das Bild wird nur invertiert.

AND Es werden nur dann Bildpunkte übertragen, wenn unter den zu übertragenden Bildpunkten bereits Bildpunkte auf dem Schirm existieren. Das vorhandene Bild wird also durch ein logisches UND mit dem zu übertragenden Bild verknüpft.

OR Es wird einem vorhandenen Bild das zu übertragende Bild überlagert (logisch mit ODER verknüpft).

XOR Es werden im existierenden Bild dort die Bildpunkte invertiert, wo im zu übertragenden Bild Bildpunkte existieren. Dadurch kann ein Objekt über den Bildschirm bewegt werden, ohne den Hintergrund zu löschen. XOR ist der voreingestellte Modus.

Die Bewegung eines Bildes kann folgendermaßen erreicht werden:

1. Das Bild wird wird mit PUT im Modus XOR auf den Bildschirm gebracht.

2. Eine veränderte Bildschirmposition wird errechnet.

3. Das Bild wird erneut mit PUT im Modus XOR an die alte Stelle gebracht, wodurch es dort gelöscht wird.

4. Schritt 1 wird mit der neuen Bildposition wiederholt.

Der Bildschirmhintergrund bleibt dabei unverändert. Bewegungen mit verändertem Hintergrund können auch mit dem Modus PSET erreicht werden. Hier sind allerdings zwei Felder für die Daten vor und nach der Veränderung erforderlich.

Anmerkungen:

- Paßt das im Feld gespeicherte Bild nicht auf den Schirm, wird die Fehlermeldung

  ```
  Illegal function call
  ```

  (unerlaubter Funktionsaufruf) angezeigt.

RANDOMIZE–Anweisung

Formate: **RANDOMIZE** [*n*]
RANDOMIZE TIMER

Bedeutung: Der Zufallszahlengenerator wird mit einem neuen Anfangswert gestartet.

n Ein beliebiger numerischer Ausdruck, dessen Wert den Ausgangswert für die Zufallszahlenerzeugung festlegt. Wird ***n*** weggelassen, fordert Amiga Basic mit der Anzeige

```
Random number seed (-32768 to 32767)?
```

die Eingabe eines Anfangswertes, ehe der Zufallszahlengenerator gestartet wird.

Wird die Anweisung RANDOMIZE TIMER ausgeführt, so wird für jeden Programmlauf, der mit RUN gestartet wird, eine neue Zufallszahlenreihe erzeugt, ohne daß ein Anfangswert angefordert wird. Der Anfangswert wird vielmehr intern mit der TIMER-Funktion erzeugt.

Beispiel:

```
RANDOMIZE
FOR I=1 TO 3
    PRINT RND
NEXT
```

ergibt nach der Angabe von z.B. 3 die Anzeige

```
Random number seed(-32768 to 32767)? 3
 .2633472681045532
 .8897488713264465
 .9760591983795166
Ok
```

Anmerkungen:

- Für jeden Programmdurchlauf werden dieselben Zufallszahlen ausgegeben (s.a. RND–Funktion), wenn kein neuer Anfangswert definiert wird.

READ–Anweisung

Format: **READ** *Var*[,*Var*]...

Bedeutung: Es werden Daten aus DATA–Anweisungszeilen gelesen und Variablen zugewiesen (s.a. DATA–Anweisung).

Var Numerische, Zeichenketten–Variable oder Feldelemente, denen entsprechende Werte aus DATA–Anweisungen zugeordnet werden.

Beispiel: Siehe Beispiel bei DATA–Anweisung.

Anmerkungen:

- Die READ–Anweisung kann nur aus DATA–Zeilen Daten lesen, die ohne Veränderung den angegebenen Variablen zugewiesen werden. Deshalb müssen die gelesenen Werte im Typ mit den bezogenen Variablen übereinstimmen. Andernfalls wird die Fehlermeldung

  ```
  Type mismatch
  ```

  (keine Typübereinstimmung)

  angezeigt.

- Mit der READ–Anweisung werden die in einer oder mehreren DATA–Zeilen gespeicherten Daten in Reihenfolge gelesen.

- Wurden bereits alle Daten gelesen, so erzeugt eine weitere READ–Anweisung die Fehlermeldung

  ```
  Out of DATA
  ```

  (zu wenig Daten), es sei denn, der Lesezeiger wurde vorher mit der RESTORE–Anweisung (s. dort) zurückgesetzt.

REM–Anweisung

Format: **REM *Kommentartext***

Bedeutung: Es können einzelne Kommentare oder komplette Kommentarzeilen in ein Programm eingefügt werden.

Kommentartext Eine beliebige Zeichenfolge.

Beispiel:

```
REM Addition der Zahlen 1 bis 10
    SUMME=0: REM Summe rücksetzen
    FOR I=1 TO 9
        SUMME=SUMME+1
    NEXT
    PRINT SUMME 'Ergebnis anzeigen
```

Anmerkungen:

- REM–Anweisungen werden nicht ausgeführt, wohl aber gelistet. Sie verlängern aber die Programmlaufzeit.

- REM–Anweisungen müssen als letzte Anweisung in einer Programmzeile stehen, wenn davor ausführbare Amiga Basic–Anweisungen in derselben Zeile stehen.

- Wird mit GOTO oder GOSUB zu einer Zeile, in der nur eine REM–Anweisung steht, verzweigt, so wird das Programm mit der nächsten auf die REM–Anweisung folgenden Zeile, die keine REM–Anweisung mehr enthält, fortgesetzt.

- Kommentare können auch mit vorangestelltem Apostroph (') in BASIC–Programmzeilen eingefügt werden (s. Beispiel). Allerdings müssen sie auch hier am Ende der Zeile stehen und dürfen **nicht** in DATA–Zeilen verwendet werden!
  Den Apostroph erhalten Sie auf einer deutschen Tastatur durch die Tastenkombination ALT-Shift-Ä.

- Kommt in einem Kommentartext ein kleines "ü" vor, ist er mit einem Gänsefüßchen einzuleiten.

RESTORE–Anweisung

Format: **RESTORE** [***Marke***]

Bedeutung: Der Lesezeiger der READ–Anweisung (s. dort) wird auf das erste DATA–Element der ersten oder einer bestimmten DATA–Zeile im Programm gestellt, um DATA–Elemente wiederholt zu lesen.

Marke Eine gültige Programmzeilennummer oder alphanumerische Sprungmarke, bei der eine DATA–Anweisung stehen muß.

Beispiel:

```
Daten:
   DATA 1,2,3,4,5,6
   READ U,V,W
   RESTORE Daten
   READ X,Y,Z
   PRINT U;V;W;X;Y;Z
```

ergibt die Anzeige

```
1 2 3 1 2 3
Ok
```

Durch die Anweisung RESTORE Daten wird, nachdem die ersten drei DATA–Elemente gelesen wurden, der Lesezeiger wieder auf den Anfang der DATA–Zeile bei der Marke Daten: gestellt. Die nachfolgende READ–Anweisung liest wieder die ersten drei DATA–Elemente.

RESUME–Anweisung

Formate: **RESUME** [0]
RESUME NEXT
RESUME ***Marke***

Bedeutung: Das Programm wird nach Ausführen einer Fehlerbearbeitungsroutine (s. ON ERROR–Anweisung) an einer definierten Stelle fortgesetzt.

RESUME oder **RESUME 0** Das unterbrochene Programm wird mit der Anweisung, die den Fehler verursacht hat, fortgesetzt.

RESUME NEXT Das unterbrochene Programm wird mit der Anweisung, die der fehlerverursachenden Anweisung folgt, fortgesetzt.

RESUME ***Marke*** Das unterbrochene Programm wird mit der durch ***Marke*** definierten Zeile fortgesetzt.

Beispiel:

```
ON ERROR GOTO Fehler
    .
    .
    .
Fehler:
IF ERR=61 THEN RESUME DiskWechsel
```

Falls die Fehlerbehandlungsroutine bei Zeile Fehler: den Fehler "Disk full" (Disk voll) diagnostiziert, soll das Programm mit der Zeile DiskWechsel: fortgesetzt werden.

Anmerkungen:

- Wenn die RESUME–Anweisung außerhalb einer Fehlerbearbeitungsroutine steht, wird die Fehlermeldung

  ```
  RESUME without error
  ```

  (RESUME ohne Fehler) angezeigt.

RETURN–Anweisung

Format: **RETURN** [***Marke***]

Bedeutung: Das Programm verzweigt nach der Ausführung einer Subroutine entweder zu der Anweisung zurück, die dem Subroutinenaufruf folgt, oder zu einer angegebenen Zeile.

Marke Eine gültige Programmzeilennummer oder alphanumerische Sprungmarke, zu der nach Abarbeitung einer nichtlokalen Ereignisunterbrechungsroutine (s.a. ON BREAK–, ON COLLISION–, ON MENU–, ON MOUSE–, ON TIMER–Anweisungen) verzweigt werden soll, damit nicht nach der aufrufenden ON *Ereignis* GOSUB–Anweisung weitergearbeitet zu werden braucht.

Anmerkungen:

- Bei der Angabe einer Zeilennummer in der RETURN–Anweisung ist mit Vorsicht vorzugehen, da alle zur Zeit der Unterbrechung aktivierten GOSUB–, WHILE– und FOR–Anweisungen auch hinterher noch aktiv bleiben.

RIGHT$–Funktion

Format: ***v$*=RIGHT$(*x$*,*n*)**

Bedeutung: Der rechte Teil der spezifizierten Zeichenkette wird in einer wählbaren Länge übergeben.

x$ Beliebiger Zeichenkettenausdruck.

n Beliebiger numerischer Ausdruck, dessen ganzzahliger Wert zwischen 0 und 32767 liegen muß.

Beispiel:

```
C$="Commodore Bueromaschinen"
PRINT RIGHT$(C$,14)
```

ergibt die Anzeige

```
Bueromaschinen
Ok
```

Die letzten 14 Zeichen der Zeichenkette C$ werden im aktuellen Ausgabefenster angezeigt.

Anmerkungen:

- Ist ***n*** größer als die Länge von ***x$***, so wird ***x$*** vollständig übergeben.
- Bei ***n*** = 0 wird eine Leer–Zeichenkette (Länge 0) übergeben.
- Weitere teilkettenbildende Funktionen sind MID$ und LEFT$ (s. dort).

RND–Funktion

Format: ***v*= RND[*(x)*]**

Bedeutung: Es wird eine Zufallszahl im Bereich zwischen 0 und 1 übergeben.

x Ein beliebiger numerischer Ausdruck, dessen Wert die übergebene Zufallszahl folgendermaßen bestimmt.

x	Wirkung
fehlt	Die nächste Zufallszahl aus einer Folge, die der Interpreter durch einen numerischen Algorithmus ermittelt, wird übergeben.
positiv	Wie oben.
negativ	Der Anfangswert für den Zufallszahlengenerator wird abhängig von ***x*** neu gesetzt, wodurch für gleiche negative ***x***–Werte immer dieselbe Folge erzeugt wird.
Null	Die letzte erzeugte Zufallszahl wird noch einmal übergeben.

Beispiel:

```
FOR i=1 TO 3:a=RND(i)
   PRINT a;                              'X>0
NEXT:PRINT
x=RND(-6)                                'X<0
FOR i=1 TO 3:a=RND(i)
   PRINT a;                              'X>0
NEXT:PRINT
RANDOMIZE 520 'neuer Anfangs-Wert
x=RND(-6)                                'X<0
FOR i=1 TO 3:a=RND(i)
   PRINT a;                              'wie X>0
NEXT:PRINT
PRINT RND(0)
```

ergibt die Anzeige

```
.1213501 .651861 .8688611
6.296182E-02 .7981562 .9540843

6.296182E-02 .7981562 .9540843
.9540842771530151
Ok
```

Anmerkungen:

- Wird der Anfangswert für den Zufallszahlengenerator nicht neu gesetzt (s. RANDOMIZE–Anweisung), so wird bei jedem Programmdurchlauf dieselbe Zufallszahlenfolge erzeugt.

- Das Neusetzen des Anfangswertes kann neben der RANDOMIZE–Anweisung auch durch die RND–Funktion mit negativem Argument erfolgen. Gleiche negative Argumente führen dann zu gleichen Zufallszahlenfolgen, die nicht durch die RANDOMIZE–Anweisung beeinflußbar sind.

- Ganze Zufallszahlen im Bereich 0 bis n können mit der Formel

  INT(RND*(n+1))

  erzeugt werden.

RSET–Anweisung

Format: **RSET *Zeichenkettenvariable = x$***

Bedeutung: Es werden in Vorbereitung für die PUT–Anweisung für Dateien Daten rechtsbündig in einen Dateipuffer für wahlfreien Zugriff übertragen.

Zeichenkettenvariable Der Name einer Zeichenkettenvariablen, die in einer FIELD–Anweisung als Datenfeld definiert wurde.

x$ Beliebiger Zeichenkettenausdruck, dessen Wert die Daten repräsentiert, die in dem durch die Zeichenkettenvariable bestimmten Feld rechtsbündig abgelegt werden sollen.

Beispiel:

```
RSET Y$=MKS$(NUM)
```

Der numerische Wert NUM wird bei der rechtsbündigen Speicherung im Datenfeld Y$ als Zeichenkette interpretiert (s. Anmerkungen und MKS$–Funktion).

Anmerkungen:

- Ist die Länge der zu speichernden Daten kleiner als das definierte Feld, so wird das Feld links mit Leerstellen aufgefüllt. Ist sie größer, wird rechts abgeschnitten.
- Numerische Werte müssen vor der Speicherung mit RSET oder LSET (s. dort) durch die Funktionen MKI$ (s. dort) als Zeichenketten interpretiert und behandelt werden.
- In LSET– und RSET–Anweisungen können auch andere als durch FIELD–Anweisungen definierte Datenfelder verwendet werden (z.B. für Druckformaterstellung).
- Weitere Informationen zum Datenaustausch mit Dateien für wahlfreien Zugriff sind im Kapitel 5 zu finden.

RUN–Befehl

Formate: **RUN** [***Marke***]
RUN ***Dateiangabe***[,**R**]

Bedeutung: Ein Amiga Basic–Programm, das entweder im Hauptspeicher resident ist oder von Disk geladen wird, wird gestartet.

Marke Eine gültige Programmzeilennummer oder alphanumerische Sprungmarke, bei der das Programm gestartet werden soll.

Dateiangabe Ein Zeichenkettenausdruck für eine Dateispezifikation, wie in Kapitel 5.2 beschrieben.

R Der Parameter **R** verhindert, wenn er angegeben ist, daß vor dem Programmstart ggf. geöffnete Dateien geschlossen werden.

Beispiel 1:

```
FOR I=1 TO 2
    PRINT I↑2;
NEXT
Hallo:
    PRINT "HALLO !"
```

Ergibt bei der Eingabe von RUN

```
 1  4 HALLO !
Ok
```

und bei RUN Hallo

```
HALLO !
Ok
```

Beispiel 2:

```
RUN "PROG1",R
```

Das Programm PROG1 wird von der Diskette im aktuellen Laufwerk geladen und gestartet. Alle vorher ggf. geöffneten Dateien bleiben offen.

Anmerkungen:

- Der Befehl RUN im Direktmodus im Ausgabefenster eingegeben, hat dieselbe Wirkung wie das Wählen von **Start** aus dem **Run**–Menü.

- Wird keine ***Marke*** angegeben, wird das Programm mit der ersten ausführbaren Zeile gestartet.

- Wird im RUN–Befehl eine Datei spezifiziert, so wird vor dem Laden, falls ein anderes Programm im Hauptspeicher ist, dem Anwender durch ein Kommunikationsfenster die Möglichkeit gegeben, dieses Programm zu speichern, da der Programmspeicher vor dem Laden gelöscht wird. Der Dateiname muß derselbe sein, unter dem das zu ladende Programm ursprünglich gespeichert wurde. Außerdem werden alle ggf. geöffneten Dateien geschlossen, es sei denn, der Parameter **R** wurde angegeben. (s.a. Kapitel 5).

SADD–Funktion

Format: ***v*=SADD(*Zeichenkettenausdruck*)**

Bedeutung: Liefert die aktuelle Speicheradresse des ersten Zeichens eines angegebenen Zeichenkettenausdruckes.

Zeichenkettenausdruck Ein beliebiger Zeichenkettenausdruck, dessen Anfangsadresse übergeben werden soll.

Beispiel:

```
CALL Anzeige(SADD("Wieviel"+chr$(0))
```

Anmerkungen:

- Die übergebene Adresse ist nur solange gültig, wie keine neue Zeichenkettenzuweisung im Programm erfolgt, da Amiga Basic seinen Zeichenkettenspeicher dynamisch verwaltet.
- Die Verwendung der Funktion VARPTR (s. dort) sollte bei Zeichenkettenvariablen vermieden werden, da sich das Format der Zeichenkettendeskriptoren, deren Adresse durch VARPTR (*Zeichenkette*) übergeben wird, zukünftig ändern kann.

SAVE–Befehl

Format: **SAVE** [***Dateiangabe***][,**A**][,**P**][,**B**]

Bedeutung: Ein im Hauptspeicher residentes Amiga Basic–Programm wird in eine Programmdatei in wählbarem Format auf Disk gespeichert.

Dateiangabe Ein Zeichenkettenausdruck für eine Dateispezifikation, wie in Kapitel 5.2 beschrieben. SAVE ohne Dateiangabe fordert den Anwender in einem Kommunikationsfenster zur Eingabe einer Dateiangabe auf.

A Das Programm wird zeilenweise in Form von ASCII–Zeichenketten gespeichert (wie LIST auf dem Bildschirm). Wenn **A** nicht angegeben oder wenn **B** angegeben ist, wird das Programm binär gespeichert.

P Das Programm wird in einem geschützten Format gespeichert.

B Das Programm wird binär gespeichert.

Beispiel 1:

```
SAVE "ADRESS"
```

Das im Hauptspeicher befindliche Programm wird binär in der Datei ADRESS auf die Disk im aktuellen Laufwerk gespeichert.

Beispiel 2:

```
SAVE "DF1:SPIEL",A
```

Das im Hauptspeicher befindliche Programm wird im ASCII–Format in die Datei SPIEL auf die Diskette im Laufwerk DF1: gespeichert.

Beispiel 3:

```
SAVE "LOHN.FIB",P
```

Das hauptspeicherresidente Programm wird als geschützte Datei unter dem Namen LOHN.FIB auf Diskette im aktuellen Laufwerk gespeichert.

Anmerkungen:

- Unter dem spezifizierten Namen im aktuellen Verzeichnis bereits existierende Dateien werden durch SAVE überschrieben.
- Fehlt die Laufwerksangabe, so wird auf die Disk (Diskette oder Festplatte) im aktuellen Laufwerk gespeichert.
- Im ASCII–Format gespeicherte Programme können wie sequentielle Datendateien gelesen werden.
- Der MERGE–Befehl (s. dort) verlangt im ASCII–Format gespeicherte Programme.
- Im geschützten Format gespeicherte Programme können weder verändert noch gelistet werden. Ein Versuch erzeugt die Fehlermeldung

  ```
  Illegal function call
  ```

  (unerlaubter Funktionsaufruf). Das geschützte Format kann nicht mehr rückgängig gemacht werden.
- Im Diskettenverzeichnis werden Programme im geschützten oder im ASCII–Format nicht besonders gekennzeichnet.

SAY–Anweisung

Format: **SAY *Zeichenkette*[,*Modus*]**

Gibt eine angegebene Zeichenkette von Phonem–Codes in verständlicher Sprache beliebiger Nationalität über einen wählbaren Tonkanal aus.

Zeichenkette Ein Zeichenkettenausdruck (Variable oder Konstante), der eine Folge von Phonem–Codes enthält (Phoneme sind Spracheinheiten, aus denen sich Silben und Wörter der gesprochenen Sprache zusammensetzen. Siehe Anmerkungen).

Modus Ein Ganzzahlfeld mit mindestens 9 Elementen, das in codierter Form die Charakteristiken der auszugebenden Sprache in vorgeschriebener Reihenfolge enthält. Folgende Codes sind in den einzelnen Feldelementen definierbar:

Element	Beschreibung
0	Grundfrequenz in Hertz. Es können Werte zwischen 65 und 320 angegeben werden. Voreingestellt ist ein Wert von 110 (normale männliche Sprech–Stimme).
1	Modulation. Es können zwei Werte angegeben werden: 0 Silben–Modulation und –Betonung (Voreinstellung). 1 Montone Sprache (roboterhaft).

Element	Beschreibung
2	Sprechgeschwindigkeit in Worten pro Minute. Es können Werte zwischen 40 und 400 angegeben werden. Voreingestellt sind 150 Worte/min.

3 Geschlecht des Sprechers. Es können zwei Werte angegeben werden:

0 Männliche Stimme (Voreinstellung).
1 Weibliche Stimme.

4 Dieses Element steuert die Änderungen in der Stimmqualität. Es können Werte zwischen 5000 (tief und grollend) und 28000 (hoch und quäkig) angegeben werden. Voreingestellt ist 22200.

5 Lautstärke. Es können Werte zwischen 0 (kein Ton) und 65 (größte Lautstärke) gewählt werden. Voreingestellt ist 65.

Element Beschreibung

6 Ton–Kanal für die Sprachausgabe. Kanäle 0 und 3 gehen auf den linken, Kanäle 1 und 2 auf den rechten Ton–Anschluß. Es können mit folgenden Werten Kanalzuordnungen vorgenommen werden:

Wert	Kanal
0	Kanal 0
1	Kanal 1
2	Kanal 2
3	Kanal 3
4	Kanäle 0 und 1
5	Kanäle 0 und 2
6	Kanäle 3 und 1
7	Kanäle 3 und 2
8	jeder verfügbare linke Kanal
9	jeder verfügbare rechte Kanal
10	jedes verfügbare Paar aus rechtem und linkem Kanal (Voreinstellung)
11	jeder verfügbare Kanal

Element	Beschreibung
7	Synchronisationsmodus. Es können zwei Werte angegeben werden: 0 Synchrone Sprachausgabe. Amiga Basic wartet auf die Beendigung der aktuell bearbeiteten SAY–Anweisung, ehe weitere Anweisungen bearbeitet werden (Voreinstellung). 1 Asynchrone Sprachausgabe. Amiga Basic startet die Berabeitung der aktuellen SAY–Anweisung im Hintergrund und fährt dann mit der Bearbeitung weiterer Anweisungen fort.
8	Steuerung der Bearbeitung von aufeinanderfolgenden SAY–Anweisungen bei asynchroner Sprachausgabe (***Modus*(7)** = 1). Es können drei Werte angegeben werden: 0 Normale Verarbeitung. Amiga Basic beendet die Berabeitung der aktuellen SAY–Anweisung, ehe die nächste SAY–Anweisung begonnen wird (Voreinstellung). 1 Die Bearbeitung der Sprachausgabe wird abgebrochen. 2 Die Berarbeitung der aktuellen SAY–Anweisung wird abgebrochen und es wird die darauffolgende SAY–Anweisung bearbeitet.

Beispiel: Im Anhang H ist an einem ausführlichen Beispiel die Verwendung der SAY–Anweisung für die Ausgabe deutscher Sprache beschrieben.

Anmerkungen:

- Wird für ***Modus*** kein Ganzzahlfeld angegeben, so wird der Fehler

  ```
  Type mismatch
  ```

  (fehlende Typübereinstimmung) angezeigt.

- Wird ***Modus*** nicht angegeben, so gelten die oben angegeben Voreinstellungen.

- Sie können Phonem–Codefolgen mit Hilfe der TRANSLATE$–Funktion (s. dort, nur für angloamerikanische Sprache) oder mit den im Anhang H beschriebenen Regeln erzeugen.

SCREEN–Anweisung

Formate: **SCREEN *n,Breite,Höhe,Tiefe,Modus***
SCREEN CLOSE *n*

Bedeutung: Definiert die Attribute (Dimensionen, Anzahl darstellbarer Farben, Auflösung) für einen neuen Bildschirm oder schließt einen Bildschirm.

n Ein ganzzahliger Ausdruck, dessen Wert zwischen 1 und 4 liegen muß, und der als Kennung für den neuen Schirm gilt. Diese Kennung wird in der WINDOW–Anweisung (s. dort) verwendet, um anzugeben, in welchem Bildschirm ein spezifiziertes Fenster liegen soll.

Breite Ein ganzzahliger Ausdruck, dessen Wert zwischen 1 und 640 liegen muß, und der die Breite des Bildschirmes in Bildpunkten angibt.

Höhe Ein ganzzahliger Ausdruck, dessen Wert zwischen 1 und 400 liegen muß, und der die Höhe des Bildschirmes in Bildpunkten angibt.

Tiefe Ein ganzzahliger Ausdruck, dessen Wert zwischen 1 und 5 liegen muß, und der die Anzahl der Bit–Ebenen für den Bildschirm festlegt. Die Anzahl der Bit–Ebenen bestimmt die Anzahl der auf dem Bildschirm darstellbaren Farben nach folgender Tabelle:

Tiefe	**Anzahl Farben**
1	2
2	4
3	8
4	16
5	32

Modus Ein ganzzahliger Ausdruck, dessen Wert zwischen 1 und 4 liegen muß, und der die Auflösung und die Bildschirmdarstellung nach folgender Tabelle festlegt:

Modus	**Bildschirm**
1	320 Bildpunkte/Zeile bei 200 Zeilen (ohne Zeilensprungverfahren).
2	640 Bildpunkte/Zeile bei 200 Zeilen (ohne Zeilensprungverfahren).
3	320 Bildpunkte/Zeile bei 400 Zeilen (mit Zeilensprungverfahren).
4	640 Bildpunkte/Zeile bei 400 Zeilen (mit Zeilensprungverfahren).

SCREEN CLOSE *n* schließt den angegebenen Bildschirm und gibt den dafür reservierten Bit–Speicher wieder frei.

Beispiel:

```
SCREEN 1,320,200,5,1
WINDOW 2,"Zeilen",(10,10)-(270,170),15,1
```

Es wird ein Bildschirm mit der Kennung 1, niedriger Auflösung mit 320x200 Bildpunkten, einer Tiefe von 5 für die Darstellung von 32 Farben ohne Zeilensprungverfahren definiert, in dem ein Fenster mit der Kennung 2 und dem Titel **Zeilen** zwischen den Koordinaten 10,10 für die obere linke Ecke und 270,170 für die untere rechte Ecke geöffnet wird. Das Fenster kann mit Hilfe der Maus vergrößert oder verkleinert, gezogen, in den Hintergrund oder Vordergrund gestellt sowie geschlossen werden (s.a. WINDOW–Anweisung).

Anmerkungen:

- Jede SCREEN–Anweisung reserviert für den spezifizierten Bildschirm einen Bit–Speicher, dessen Größe von den Schirmdimensionen abhängt.

- Ist der Amiga an ein Fernsehgrät angeschlossen, so wird üblicherweise niedrige Auflösung (320 Bildpunkte/Zeile) gewählt, bei Monochrom– oder RGB–Monitoren dagegen hohe Auflösung.

- Beim Zeilensprungverfahren (auch Zwischenzeilenabtastung genannt) wird die Anzahl horizontaler Bildzeilen verdoppelt, was aber unter Umständen zu einem leichten Flimmern des Bildes führen kann.

SCROLL–Anweisung

Format: **SCROLL (*x1,y1*)–(*x2,y2*),*deltax,deltay***

Bedeutung: Rollt einen definierten, rechteckigen Bereich im aktuellen Ausgabefenster in eine wählbare Richtung.

x1,y1 Absolute Bildschirmkoordinaten der linken oberen Ecke des zu rollenden Bereiches.

x2,y2 Absolute Bildschirmkoordinaten der rechten unteren Ecke des zu rollenden Bereiches.

deltax Ein numerischer Ausdruck, dessen ganzzahliger Wert die Anzahl der Bildpunkte angibt, um die nach rechts (***deltax*** positiv) oder links (***deltax*** negativ) gerollt werden soll.

deltay Ein numerischer Ausdruck, dessen ganzzahliger Wert die Anzahl der Bildpunkte angibt, um die nach unten (***deltay*** positiv) oder oben (***deltay*** negativ) gerollt werden soll.

Anmerkungen:

- Die SCROLL–Anweisung ist am effektivsten, wenn das zu rollende Bild kleiner als das definierte Rechteck ist, und in den betroffenen Bildschirmbereichen im Hintergrund keine Darstellungen vorhanden sind.

SGN–Funktion

Format: ***v*=SGN(*x*)**

Bedeutung: Es wird das Vorzeichen des Wertes eines beliebigen numerischen Ausdruckes in Form einer Konstanten übergeben.

x Ein beliebiger numerischer Ausdruck.

Es gilt:

für $x<0$: SGN(x) = −1
für $x=0$: SGN(x) = 0
für $x>0$: SGN(x) = 1

Beispiel:

```
IF SGN(A) THEN NichtNull
PRINT "A ist Null":END
    .
    .
    .
NichtNull:
    .
    .
    .
```

Das Programm verzweigt nur nach Zeile NichtNull:, wenn der Wert von A positiv oder negativ ist.

SHARED–Anweisung

Format: **SHARED** ***Variable***[,***Variable***]. . . .

Bedeutung: Definiert innerhalb eines Unterprogramms die angegebenen Variablen als globale Variablen.

Variable Eine beliebige Variable oder Feldvariable, die als global für Haupt– und Unterprogramm definiert werden soll. Feldvariablen müssen ein Paar runde Klammern () folgen.

Beispiel: Siehe Kapitel 6.1.

Anmerkungen:

- Globale Variable sind Variablen, deren Werte sich im Hauptprogramm analog ändern, wenn sie im Unterprogramm geändert werden und umgekehrt. Normalerweise sind alle in Unterprogrammen deklarierten Variablen zunächst lokal, also nur für das betreffende Unterprogramm gültig.

- Die SHARED–Anweisung darf nur in Unterprogrammen verwendet werden, kann dort aber mehrfach vorkommen.

- Es ist guter Programmierstil, alle SHARED–Anweisungen eines Unterprogrammes an dessen Anfang zu stellen.

- Die Dimensionierung globaler Felder mit der DIM SHARED-Anweisung (s. dort) muß jedoch im Hauptprogramm erfolgen.

SIN–Funktion

Format: ***v*=SIN(*x*)**

Bedeutung: Es wird der Sinus des Wertes des numerischen Ausdruckes ***x*** berechnet und übergeben. Ist das Argument einfach genau, erfolgt die Berechnung einfach–genau, ist es doppelt–genau, so wird auch ein doppelt–genauer Wert übergeben.

x Winkel im Bogenmaß.

Beispiel:

```
PI=3.141593
WINKEL=180
RADIANS=WINKEL*PI/180
PRINT SIN(RADIANS)
```

Wird dieses Programm gestartet so ergibt sich die Anzeige

```
0
Ok
```

Um den Sinus des Winkels von 180 Grad zu ermitteln, wird der Winkel in der dritten Programmzeile ins Bogenmaß umgerechnet.

Anmerkungen:

- Weitere trigonometrische Funktionen sind COS und TAN (s. dort).
- Im Anhang E sind die transzendenten Funktionen zusammengestellt, die mit Hilfe der vorhandenen Amiga Basic-Funktionen umschrieben werden können.

SLEEP–Anweisung

Format: **SLEEP**

Bedeutung: Ein Amiga Basic–Programm wird angehalten, bis eines der folgenden Ereignisse eintritt.

- Betätigung der Auswahltaste der Maus
- Betätigung einer Tastaturtaste
- Kollision grafischer Objekte
- Auswahl eines Menü–Titels
- Ablauf einer vorgegebenen Zeit (TIMER)

Beispiel:

```
WarteTaste:
    i$=INKEY$
    IF i$="*" THEN STOP
    SLEEP
GOTO WarteTaste
```

Es wird auf das Drücken einer Taste gewartet. Nur wenn es die Taste * ist, wird das Programm abgebrochen.

SOUND–Anweisung

Formate: **SOUND *Frequenz,Dauer*[,[*Laut*][,**Kanal**]]**
SOUND WAIT
SOUND RESUME

Bedeutung: Es wird ein Ton einer wählbaren Frequenz, Dauer und Lautstärke in einer Ton–Warteschlange abgelegt, die über einen wählbaren Ton–Kanal abgespielt wird. Das Abspielen der Warteschlange kann außerdem angehalten oder fortgesetzt werden.

Frequenz Eine beliebige numerische Ganzahl– oder Festpunkt–Konstante (letztere in einfacher oder doppelter Genauigkeit), deren Wert im Bereich zwischen 20 und 15000 liegen muß und die gewünschte Frequenz in Hertz definiert. Liegt er unter 20, erzeugt Amiga Basic einen Ton von 20 Hertz, liegt er über 15000, erzeugt Amiga Basic einen Ton von 15000 Hertz (s. Anmerkungen).

Dauer Ein beliebiger numerischer Ausdruck, dessen Wert im Bereich zwischen 0 und 77 liegen muß und der die Tondauer in Zeittakten definiert. Eine Sekunde entspricht der Dauer von 18,2 Zeittakten (s. Anmerkungen).

Laut Ein numerischer Ausdruck, dessen ganzzahliger Wert zwischen 0 (kleinste Lautstärke) und 255 (größte Lautstärke) liegen muß und der die Lautstärke des Tones angibt. Voreingestellt ist hier 127.

Kanal Ein numerischer Ausdruck, dessen ganzzahliger Wert zwischen 0 und 3 liegen muß. Ein Wert von 0 oder 3 definiert für diese beiden Tonkanäle den linken, ein Wert von 1 oder 2 für diese Tonkanäle den rechten Tonausgang des Amiga. Voreingestellt ist hier der Wert 0 (Null).

SOUND WAIT Nach dieser Anweisung werden die Ergebnisse aller folgenden SOUND–Anweisungen in einer Warteschlange abgelegt, bis eine SOUND RESUME–Anweisung ausgeführt wird. Dies dient zur Synchronisation der Töne aus allen vier Amiga–Tonkanälen. Da die Ton–Warteschlange begrenzt ist,

kann das Ansammeln von zu vielen Tönen in der Warteschlange ohne SOUND RESUME–Anweisung zu der Fehleranzeige

```
Out of Memory
```

(Hauptspeicher nicht ausreichend) führen.

Beispiel:

```
FOR I=440 TO 1000 STEP 5
    SOUND I,1.5
NEXT
FOR I=1000 TO 440 STEP -5
    SOUND I,1.5
NEXT
```

Durch dieses Programm wird ein auf– und abschwellendes Glissando mittlerer Lautstärke erzeugt und von Kanal 0 über den linken Tonausgang ausgegeben.

Anmerkungen:

- Die beiden folgenden Tabellen geben die Frequenzen für zwei Oktaven vor und nach dem kleinen c sowie die verschiedenen Tempi in Abhängigkeit von den Zeittakten:

Note	Frequenz	Note	Frequenz
C	130.810	c	523.250
D	146.830	D	587.330
E	164.810	E	659.260
F	174.610	F	698.460
G	196.000	G	783.990
A	220.000	A	880.000
H	246.940	H	987.770
C	261.630	C	1046.500
D	293.660	D	1174.700
E	329.630	E	1318.500
F	349.230	F	1396.900
G	392.000	G	1568.000
A	440.000	A	1760.000
H	493.880	H	1975.500

Tempo	Takte/ Minute	Einheiten/ Takt
Larghissimo	40– 60	28.13–18.75
Largo	60– 66	18.75–17.05
Larghetto		
Grave		
Lento		
Adagio	66– 76	17.05–14.8
Adagietto		
Andante	76–108	14.8–10.42
Andantino		
Moderato	108–120	10.42–9.38
Allegretto		
Allegro	120–168	9.38– 6.7
Vivace		
Veloce		
Presto		
Prestissimo	168–208	6.7 – 5.41

- Höhere oder tiefere Noten können durch Verdoppelung oder Halbierung der Frequenz der betreffenden Note in der nächsttieferen oder nächsthöheren Oktave angenähert werden.

- Bei aufeinanderfolgenden SOUND–Anweisungen wird eine Folgeanweisung erst ausgeführt, wenn die vorhergehende Anweisung mit ihrer vollen Dauer abgearbeitet ist.

- Die Vielseitigkeit der SOUND–Anweisung kann mit Hilfe der WAVE–Anweisung (s. dort), mit der die Tonwellenform festgelegt werden kann, erheblich erweitert werden.

SPACE$–Funktion

Format: ***v$*=SPACE$(*n*)**

Bedeutung: Es wird eine Zeichenkette, die aus einer wählbaren Anzahl von Leerstellen besteht, übergeben.

n Ein ganzzahliger Ausdruck, dessen Wert im Bereich zwischen 0 und 32767 liegen muß.

Beispiel:

```
A$="ZWISCHEN"+SPACE$(4)+"RAUM"
PRINT A$
ZWISCHEN    RAUM
Ok
```

Anmerkungen:

- Siehe auch SPC–Funktion.

SPC–Funktion

Format: **SPC(*n*)**

Bedeutung: Es wird eine wählbare Anzahl von Leerstellen auf dem Ausgabegerät (Bildschirm, Drucker, sequentielle Datei) ausgegeben.

n Ein ganzzahliger Ausdruck, dessen Wert im Bereich zwischen 0 und 255 liegen muß.

Beispiel:

```
PRINT "ZWISCHEN"SPC(4)"RAUM"
```

ergibt im Ausgabefenster

```
ZWISCHEN    RAUM
Ok
```

Steht die SPC–Funktion am Ende einer Liste von Ausdrücken in einer PRINT–Anweisung, so wird hier **kein** Wagenrücklauf/Zeilenvorschub erzeugt.

Anmerkungen:

- Die SPC-Funktion darf nur in Verbindung mit PRINT-, PRINT #- und LPRINT-Anweisungen verwendet werden.
- Siehe auch SPACE$–, PTAB– und TAB–Funktionen.

SQR–Funktion

Format: ***v*=SQR(*x*)**

Bedeutung: Es wird die Quadratwurzel des Wertes eines numerischen Ausdruckes je nach Genauigkeit des Argumentes mit einfacher oder doppelter Genauigkeit berechnet. Der Wert muß größer oder gleich Null sein.

x Beliebiger numerischer Ausdruck mit positivem Wert.

Beispiel:

```
FOR I=0 TO 9 STEP 3
    PRINT I,SQR(I)
NEXT
```

ergibt im Ausgabefenster

```
0             0
3             1.732051
6             2.44949
9             3
Ok
```

STICK–Funktion

Format: ***v*=STICK(*n*)**

Bedeutung: Liefert den Zustand eines wählbaren, angeschlossenen Joysticks (Spielpult, Steuerknüppel).

n Ein ganzzahliger Ausdruck, dessen Wert zwischen 0 und 3 liegen muß, und der festlegt, welche Information von welchem Joystick abgefragt werden soll:

n	**Joystick und Richtung**
0	Joystick A in X–Richtung
1	Joystick A in Y–Richtung
2	Joystick B in X–Richtung
3	Joystick B in Y–Richtung

STICK(*n*) liefert einen der folgenden Werte als Richtungsindikator:

1 Bewegung nach oben oder rechts
0 Jystick wird nicht betätigt
–1 Bewegung nach unten oder links

STOP–Anweisung

Format: **STOP**

Bedeutung: Die Programmausführung wird unterbrochen, und Amiga Basic kehrt auf die Befehlsebene (Direktmodus) zurück. Das Programm kann durch Eingabe des CONT–Befehls (s. dort) fortgesetzt werden.

Beispiel:

```
PI=3.141593:D=10
PRINT "Kreisflaeche ist";PI*D↑2/4
STOP
PRINT "Kreisumfang ist";PI*D
```

ergibt im Ausgabefenster:

```
Kreisflaeche ist 78.53983
Ok
```

CONT (wird im Direktmodus eingegeben)

```
Kreisumfang ist 31.41593
Ok
```

Anmerkungen:

- Die STOP–Anweisung kann an beliebigen Stellen in ein Programm zum Testen eingefügt werden, um z.B. den Wert von bestimmten Variablen zu überprüfen.
- Die STOP–Anweisung schließt im Gegensatz zur END–Anweisung (s. dort) keine geöffneten Dateien.
- Die STOP–Anweisung hat dieselbe Wirkung wie das Wählen von **Stop** aus dem **Run**–Menü.

STR$–Funktion

Format: ***v$*=STR$(*x*)**

Bedeutung: Es wird der Wert eines numerischen Ausdrucks als Zeichenkette übergeben.

x Ein beliebiger numerischer Ausdruck.

Beispiel:

```
A=12.45:PRINT STR$(A),LEN(STR$(A))
 12.45          6
Ok
```

Anmerkungen:

- Der Zeichenkette eines positiven Wertes wird immer anstelle des Vorzeichens eine Leerstelle vorangestellt.
- Die Umkehrfunktion von STR$ ist die VAL–Funktion (s. dort).

STRIG–Funktion

Format: ***v*= STRIG(*n*)**

Bedeutung: Es wird je nach Argument ***n*** der Status des Feuerknopfes eines wählbaren Joysticks übergeben.

n Ein ganzzahliger Ausdruck, dessen Wert im Bereich zwischen 0 und 3 liegen muß, und der folgende Bedeutung hat:

0 Es wird 1 übergeben, falls der Knopf von Joystick A seit der letzten Abfrage mit STRIG(0) gedrückt wurde, sonst 0.

1 Es wird 1 übergeben, falls der Knopf von Joystick A gerade gedrückt ist, sonst 0.

2 Es wird 1 übergeben, falls der Knopf von Jystick B seit der letzten Abfrage mit STRIG(2) gedrückt wurde, sonst 0.

3 Es wird 1 übergeben, falls der Knopf von Joystick B gerade gedrückt ist, sonst 0.

STRING$–Funktion

Format: ***v$*=STRING$(*n,m*)**
***v$*=STRING$(*n,x$*)**

Bedeutung: Es wird eine Zeichenkette gleicher Zeichen wählbarer Länge ***n*** übergeben, die entweder aus Zeichen eines wählbaren ASCII-Codes ***m*** oder aus dem Anfangszeichen einer spezifizierten Zeichenkette ***x$*** besteht.

n Ganzzahliger Ausdruck, dessen Wert im Bereich zwischen 0 und 32767 liegen muß.

m Ganzzahliger Ausdruck, dessen Wert im Bereich zwischen 0 und 255 liegen muß.

x$ Ein beliebiger Zeichenkettenausdruck.

Beispiel 1:

```
u$=STRING$(9,42)
PRINT "Betrag: "u$
```

ergibt im Ausgabefenster

```
BETRAG: *********
Ok
```

Zwischen dem Text und der Variablen u$ braucht kein Semikolon angegeben zu werden, da das zweite Anführungszeichen als Trennmerkmal gilt.

Beispiel 2:

```
PRINT STRING$(5,"Xantippe")
```

ergibt im Ausgabefenster

```
XXXXX
Ok
```

SUB–Anweisung

Formate: **SUB *Name*[(*Liste form. Param.*)] STATIC**
END SUB
EXIT SUB

Bedeutung: Leitet ein Amiga Basic–Unterprogramm ein, beendet es bzw. erlaubt den Unterprogramm–Aussprung.

Name Jeder beliebige Name bis zu einer Länge von 40 Zeichen nach den Amiga Basic– Namensregeln (s. Kapitel 8.3 und 8.5). Dieser Name darf in keiner anderen SUB–Anweisung verwendet werden.

Liste form. Param. Hier dürfen einfache oder Feld–Variablen angegeben werden. Bei mehreren Einträgen sind diese durch Kommata voneinander zu trennen. Die Anzahl der Einträge wird nur durch die maximale Länge einer logischen Amiga Basic–Programmzeile (254 Zeichen) begrenzt. Werden Feld–Variablen angegeben, so muß dies durch ein Paar runder Klammern () gekennzeichnet werden. Innerhalb der Klammern kann die Anzahl der Dimensionen des Feldes, **nicht** die einzelnen Dimensionen selbst, angegeben werden.

STATIC besagt, daß alle Variablen im Unterprogramm ihre Werte zwischen zwei Unterprogrammaufrufen behalten. Die Werte von mit STATIC deklarierten Variablen können nicht vom Hauptprogramm verändert werden. Außerdem bedeutet der STATIC-Zusatz, daß das betreffende Unterprogramm nicht rekursiv ist, d.h. sich nicht selbst, noch ein anderes Unterprogramm aufrufen darf, das dann seinerseits das aufrufende Unterprogramm wieder aufruft.

END SUB bezeichnet das Ende eines Unterprogrammes und übergibt die Programmsteuerung an die Anweisung, die dem Unterprogrammaufruf im Hauptprogramm folgt.

EXIT SUB erlaubt die Beendigung eines Unterprogramms in dessen Mitte und die Übergabe der Programmsteuerung an die Anweisung, die dem Unterprogrammaufruf im Hauptprogramm folgt.

Beispiel: Siehe Kapitel 6.1

Anmerkungen:

- Ehe Amiga Basic ein Unterprogramm ausführt, werden alle damit zusammenhängenden Anweisungen formal überprüft. Werden Fehler gefunden, so wird das Unterprogramm nicht ausgeführt. Diese Fehler sind nicht mit einer Fehleroutine programmiert verarbeitbar, noch existieren Fehlercodes. Sie werden vielmehr in einem Fehler–Kommunikationsfenster angezeigt. Es gibt folgende Fehlermeldungen, die sich auf die formale Überprüfung eines Unterprogramms beziehen:

Tried to declare a SUB within a SUB

Versuch, SUB innerhalb einer SUB–Struktur zu deklarieren.

SUB already defined

SUB bereits deklariert

Missing STATIC in SUB–Statement

in SUB–Anweisung fehlt STATIC–Zusatz

EXIT SUB outside of a subprogramm

EXIT SUB–Anweisung außerhalb eines Unterprogramms

END SUB outside of a subprogramm

END SUB–Anweisung außerhalb eines Unterprogramms

SUB without an END SUB

END SUB–Anweisung fehlt in einer SUB–Struktur

SHARED outside of a subprogramm

SHARED–Anweisung außerhalb eines Unterprogramms.

- In Kapitel 6.1 wird die Verwendung von Amiga Basic–Unterprogrammen ausführlich, auch an Hand von Beispielen, erläutert.

- Siehe auch CALL– und SHARED–Anweisungen

SWAP–Anweisung

Format: **SWAP *Var1,Var2***

Bedeutung: Die Werte zweier beliebiger Variablen gleichen Typs werden miteinander vertauscht.

Var1,Var2 Zwei Namen für beliebige Variablen oder Feldelemente, die im Typ und ggf. in der Genauigkeit übereinstimmen müssen.

Beispiel:

```
X$="ROCK":Y$="NACHT"
PRINT X$Y$:SWAP X$,Y$:PRINT X$Y
```

ergibt folgende Anzeige im Ausgabefenster:

```
ROCKNACHT
NACHTROCK
Ok
```

Anmerkungen:

- Bei fehlender Typ– oder Genauigkeitsübereinstimmung wird die Fehlermeldung

  ```
  Type mismatch
  ```

  (keine Typübereinstimmung) ausgegeben.

SYSTEM–Befehl

Format: **SYSTEM**

Bedeutung: Rückkehr aus Amiga Basic zum Arbeitstisch (Workbench) oder auf die Befehlsebene (CLI) des AmigaDOS–Betriebssystems.

Anmerkungen:

- SYSTEM schließt alle offenen Dateien.
- der SYSTEM–Befehl hat dieselbe Wirkung wie die Wahl von **QUIT** im **Project**–Menü.
- Amiga Basic prüft vor der Ausführung des SYSTEMS–Befehl, ob das Programm seit der letzten Speicherung verändert wurde. Ist dies der Fall, so wird der Anwender in einem Kommunikationsfenster zu einer Entscheidung aufgefordert, ob er das Programm vorher noch speichern möchte oder nicht.

TAB–Funktion

Format: **TAB(*n*)**

Bedeutung: Die Zeile auf dem Ausgabegerät (Bildschirm, Drucker, sequentielle Datei) wird von der augenblicklichen bis zur definierten Druckposition mit Leerstellen aufgefüllt (Tabulator).

n Ein ganzzahliger Ausdruck, dessen Wert zwischen 1 und 255 liegen muß.

Beispiel:

```
DATA "BANANEN","3.99":READ A$,P$
PRINT "Obst"TAB(20)"kg-Preis":PRINT
PRINT A$TAB(20)P$

ergibt im Ausgabefenster die Anzeige
Obst                          kg-Preis
Bananen                       3.99
Ok
```

Anmerkungen:

- Die TAB–Funktion darf nur in Verbindung mit den PRINT–, PRINT #– und LPRINT–Anweisungen verwendet werden.
- Wenn die aktuelle Druckposition rechts von der durch ***n*** spezifizierten Position liegt, wird an die Position ***n*** in der nächsten Zeile tabuliert.
- Die Spalte 1 ist die äußerst linke Druckposition. Die äußerst rechte ist durch die WIDTH–Einstellung definiert.
- Steht die TAB–Funktion am Ende einer Liste von Ausdrükken, so wird kein Wagenrücklauf/Zeilenvorschub ausgeführt.
- Siehe auch PTAB– und SPC–Funktionen.

TAN–Funktion

Format: ***v*=TAN(*x*)**

Bedeutung: Es wird der Tangens des Wertes des numerischen Ausdruckes ***x*** berechnet und übergeben. Ist das Argument einfach–genau, erfolgt die Berechnung mit einfacher, ist es doppelt–genau, mit doppelter Genauigkeit.

x Winkel im Bogenmaß.

Beispiel:

```
PI=3.141593
WINKEL=45:RADIANS=WINKEL*PI/180
PRINT TAN(RADIANS)
```

ergibt im Ausgabefenster die Anzeige

```
1
Ok
```

Um den Tangens eines Winkels von 45 Grad zu berechnen, wird der Winkel zunächst ins Bogenmaß umgerechnet.

Anmerkungen:

- Tritt beim Tangens Überlauf ein, so meldet der Interpreter dies mit

  ```
  Overflow
  ```

  (Überlauf), übergibt als Ergebnis die für ihn größte mögliche Zahl mit dem entsprechenden Vorzeichen und führt das Programm fort.

- Siehe auch COS– und SIN–Funktionen.

TIME$–Systemvariable

Format: ***v$*** **= TIME$**

Bedeutung: Die laufende Zeit wird von der Systemuhr abgerufen und als Zeichenkette von 8 Bytes Länge im Format ***hh.mm.ss*** übergeben. ***hh*** liegt im Bereich 00 bis 23, ***mm*** und ***ss*** im Bereich 00 bis 59.

Beispiel:

```
100 LOCATE 1,1:PRINT TIME$:GOTO 100
```

Die laufende Zeit wird in der oberen linken Ecke des aktuellen Ausgabefensters angezeigt.

TIMER–Anweisung

Formate: **TIMER ON**
TIMER OFF
TIMER STOP

Bedeutung: Die Unterbrechungsreaktionsfähigkeit für Zeitablauf wird aktiviert oder inaktiviert.

Beispiel: Siehe ON TIMER–Anweisung für ein ausführliches Beispiel.

Anmerkungen:

- TIMER ON aktiviert die Unterbrechungsreaktionsfähigkeit für Zeitablauf. In einem Programm prüft der Interpreter dann vor jeder neuen Anweisung, ob eine definierte Zeit abgelaufen ist und verzweigt ggf. zu der in der ON Timer(*n*) GOSUB–Anweisung (s. dort) angegebenen Subroutine.
- Eine TIMER–Anweisung darf nicht vor einer ON TIMER(*n*) GOSUB–Anweisung im Programm stehen.
- TIMER OFF inaktiviert die Unterbrechungsreaktionsfähigkeit. Unterbrechungen durch Zeitablauf sind nach dieser Anweisung dann nicht mehr möglich.
- TIMER STOP inaktiviert zunächst auch die Unterbrechungsreaktionsfähigkeit. Die Unterbrechung beim Ablauf einer definierten Zeit erfolgt jedoch sofort, sobald die Anweisung TIMER ON ausgeführt wird.
- S.a. TIMER–Funktion sowie Kapitel 6.4 "Verarbeitung von Unterbrechungsereignissen".

TIMER–Funktion

Format: ***v*= TIMER**

Bedeutung: Es wird die Anzahl der Sekunden, die seit Mitternacht oder dem Wiederanlauf des Rechnersystems vergangen sind, als Wert einfacher Genauigkeit übergeben.

Anmerkungen:

- TIMER kann nicht vom Anwender gesetzt werden.
- Die TIMER–Funktion kann als Anfangswertgeber bei der RANDOMIZE–Anweisung (s. dort) verwendet werden.

TRANSLATE$–Funktion

Format: ***v$*** **= TRANSLATE$(*Text*)**

Bedeutung: Erzeugt und übergibt eine Folge von Phonemcodes für die SAY–Anweisung (s. dort) aus einer angloamerikanischen Text-Zeichenkette.

Text Ein Zeichenkettenausdruck, dessen Wert ein Text in angloamerikanischer Sprache sein muß und der als Sprache mit der SAY–Anweisung ausgegeben werden soll.

Beispiel:

```
A$=TRANSLATE$("I'm the Amiga computer")
SAY a$
```

Anmerkungen:

- TRANSLATE$ kann eine Phonem–Zeichenkette von maximal 32767 Zeichen erzeugen.
- Im Anhang H ist die Verwendung der TRANSLATE$–Funktion sowie der SAY–Anweisung ausführlich beschrieben. Anhand eines Beispielprogramms wird dort auch gezeigt, wie deutsche Sprache erzeugt werden kann.

TRON– und TROFF–Befehle

Formate: **TRON**
TROFF

Bedeutung: Der Programmablauf wird durch den Befehl TRON im List–Fenster protokolliert. Dabei wird vor der Ausführung einer Anweisung diese mit einem orangefarbenen Rechteck eingerahmt im List–Fenster angezeigt, falls das List–Fenster sichtbar ist. Der Befehl TROFF schaltet diese Programmablaufverfolgung wieder aus.

Anmerkungen:

- Die TRON– und TROFF–Befehle dienen zum schrittweisen Austesten von Programmen. Sie haben dieselbe Wirkung wie die Wahl des **Trace On/Off**–Schalters im **Run**–Menü (s.a. Kapitel 3.4).

UBOUND–Funktion

Siehe LBOUND–Funktion

UCASE$–Funktion

Format: ***v$*=UCASE$(*x$*)**

Bedeutung: Übergibt eine Kopie einer angegebenen Zeichenkette, in der alle Kleinbuchstaben in Großbuchstaben gewandelt werden.

x$ Beliebiger Zeichenkettenausdruck, der in gleicher Länge übergeben wird, nachdem vorher alle Kleinbuchstaben in Großbuchstaben gewandelt wurden.

Beispiel:

```
LINE INPUT "Weiter (j/n) ? ";a$
IF UCASE$(a$)="J" THEN Weiter
END
.
.
.
Weiter:
.
.
.
```

Auf die Abfrage könnte der Anwender sowohl mit j als auch mit J antworten. Mit Hilfe der UCASE$–Funktion können beide Fälle berücksichtigt werden.

Anmerkungen:

- UCASE$ ist vor allem hilfreich, wenn Textdaten sortiert werden sollen, da ja Klein– und Großbuchstaben gleich sortiert werden sollen, aber unterschiedliche ASCII–Codes haben. Mit UCASE$ können solche Daten für die Sortierung umgewandelt werden, ohne die Originaldaten verändern zu müssen.

VAL–Funktion

Format: ***v*=VAL(*x$*)**

Bedeutung: Der Wert eines Zeichenkettenausdruckes wird in Form des numerischen Äquivalentes übergeben.

x$ Ein beliebiger Zeichenkettenausdruck.

Beispiel 1:
```
A$="12.45":PRINT VAL(A$)
 12.45
Ok
```

Beispiel 2:
```
A$="Preis DM 8.90":PRINT VAL(A$)
 0
Ok
```

Anmerkungen:

- Beginnt der Zeichenkettenausdruck mit einem anderen Zeichen als einer Ziffer, dem Punkt (.) oder dem Plus– (+) bzw. Minus–Zeichen (–), so übergibt VAL eine Null (0).
- VAL ignoriert alle Leerstellen, Tabulatoren und Zeilenvorschubcodes in der Zeichenkette.
- Außer den oben genannten Zeichen am Anfang der Zeichenkette akzeptiert VAL noch die Buchstaben E und D in der Zeichenkette (z.B. VAL("3E–12")).
- Die Umkehrfunktion von VAL ist die STR$–Funktion (s. dort).

VARPTR–Funktion

Format: ***v*=VARPTR(*Variable*)**

Bedeutung: Die VARPTR–Funktion liefert die Hauptspeicheradresse im Bereich zwischen 0 und 16777215 des ersten Datenbytes einer beliebigen Variablen oder eines Feldvariablenelementes.

Variable Eine beliebige Variable oder ein beliebiges Feldvariablenelement, der/dem ein Wert zugewiesen sein muß.

Beispiel:

```
BX%=1024
VP=VARPTR(BX%)
AX=PEEKW(VP)
```

Zunächst wird VP die Adresse der Ganzzahlvariablen BX% zugewiesen. Die Daten, die in dieser Variablen gespeichert sind, werden dann mit PEEKW (s. dort) gelesen.

Anmerkungen:

- Bei Zeichenkettenvariablen übergibt VARPTR die Adresse der ersten Bytes des Zeichenkettendeskriptors, also der Länge der Zeichenkette. Die folgenden Bytes enthalten dann die Adresse der Zeichenkette selbst. Da sich das Deskriptorformat in zukünftigen Versionen von Amiga Basic ändern kann, ist hier besser die SADD–Funktion (s. dort) zu verwenden.

- Die VARPTR–Funktion eignet sich besonders zum Übergeben von Variablenadressen an Maschinensprache–Unterprogramme.

- Wird VARPTR für ein Feldvariablenelement verwendet, sollten alle einfachen Variablen vorher definiert sein, damit sich die Feldelementadressen nicht bei jeder neuen einfachen Variablendefinition ändern.

- Wenn ein Feld an ein Maschinensprache–Unterprogramm übergeben werden soll, wird üblicherweise z.B. mit VARPTR (A(0)) Die Anfangsadresse des ersten Elementes übergeben.

- Die Verwendung der VARPTR–Funktion in Verbindung mit Maschinensprache–Unterprogrammen ist ausführlich in Kapitel 6.2 beschrieben.

- Wurde der in einem VARPTR–Aufruf deklarierten Variablen noch kein Wert zugewiesen, so wird die Fehlermeldung

  ```
  Illegal function call
  ```

  (unerlaubter Funktionsaufruf) angezeigt.

WAVE–Anweisung

Format: **WAVE *Kanal,Definition***

Bedeutung: Definiert die Form von Tonwellen für einen angegebenen Tonkanal.

Kanal Ein numerischer Ausdruck, dessen ganzzahliger Wert im Bereich zwischen 0 und 3 liegen muß, und der den Ton–Kanal des Amiga angibt, über den die Ton–Welle mit der SOUND–Anweisung (s. dort) ausgegeben werden soll. 0 oder 3 beziehen sich auf den linken, 1 oder 2 auf den rechten Ton–Ausgang des Amiga.

Definition Entweder das Schlüsselwort SIN oder ein numerisches Feld mit mindestens 256 Elementen. Jedes Element darf einen ganzzahligen Wert zwischen –128 und 127 annehmen und definiert eine Amplitude. Die Aneinanderreihung dieser Amplitudenwerte ergibt die Tonwellenform.

Beispiel:

```
DEFINT a-z
DIM timbre
FOR i=0 TO 255
    READ timbre
NEXT
WAVE 0,SIN
WAVE 1,timbre
WAVE 2,timbre
WAVE 3,timbre
```

Anmerkungen:

- Nach der Ausführung der WAVE–Anweisung sollte mit der ERASE–Anweisung das Wellenformfeld wieder gelöscht werden, um Speicherplatz zu sparen.

WHILE . . . WEND–Anweisungen

Format:

WHILE ***Ausdruck***
.
.
Anweisungen
.
.
WEND

Bedeutung: Eine beliebige Anzahl von Anweisungen, die in beliebig vielen Programmzeilen stehen können, wird in einer Schleife so oft ausgeführt, wie ein angegebener Ausdruck logisch wahr (von Null verschieden) ist.

Ausdruck Ein beliebiger Ausdruck, dessen Wert logisch geprüft wird: Er wird in der WHILE-Anweisung ausgewertet. Ist er "wahr" (von Null verschieden), werden alle Anweisungen bis zur WEND-Anweisung ausgeführt, zu WHILE zurückgesprungen und der Ausdruck erneut überprüft. Ist der Ausdruck logisch "falsch" (Null), wird das Programm mit der auf WEND folgenden Anweisung fortgesetzt.

Beispiel:

```
' Dezimal -> Hexadezimal-Wandlung
Antwort$="J"
WHILE (UCASE$(Antwort$)="J")
    INPUT"Dezimalzahl";dez
    PRINT"Hex-Wert von"dez" ist "HEX$(dez)
    PRINT"Okt-Wert von"dez" ist "OCT$(dez)
    INPUT"Weiter (j/n) ";Antwort$
WEND
END
```

Anmerkungen:

- WHILE . . . WEND–Anweisungen können beliebig geschachtelt werden. Eine WEND–Anweisung wird vom Interpreter immer der letzten vorausgegangenen WHILE–Anweisung zugeordnet.

- Fehlt eine WHILE–Anweisung zu einer WEND–Anweisung, wird die Fehlermeldung

  `WEND without WHILE`

  (WEND ohne WHILE) angezeigt.

- Fehlt eine WEND–Anweisung zu einer WHILE–Anweisung, wird die Fehlermeldung

  `WHILE without WEND`

  (WHILE ohne WEND) angezeigt.

- **Achtung: Springen Sie niemals in eine WHILE-WEND-Struktur, ohne die einführende WHILE–Anweisung ausführen zu lassen, da dieses die Programmablaufsteuerung von Amiga Basic durcheinanderbringt.**

WIDTH–Anweisung

Formate:

Syntax 1: **WIDTH [LPRINT]** [***Breite***] [,***Druckzone***]
Syntax 2: **WIDTH#*Dateinr*** [,***Breite***][,***Druckzone***]
Syntax 3: **WIDTH *Gerät*** [,***Breite***] [,***Druckzone***]

Bedeutung: Es wird die Druckzeilenbreite für die Ausgabe an eine Ausgabeeinheit (Drucker, Bildschirm, Datenfernübertragungskanal) in Standardzeichen festgelegt. Nach der Ausgabe einer Zeile in der mit dieser Anweisung definierten Breite wird ein Wagenrücklaufcode angefügt. In den Proportional–Zeichensätzen des Amiga gilt als Standardbreite die Breite einer beliebigen Ziffer zwischen 0 und 9. Die voreingestellte Breite für den Bildschirm ist 255.

Breite Ein ganzzahliger Ausdruck, dessen Wert zwischen 0 und 255 liegen muß. Eine Breite von Null wird als 1 interpretiert.

Druckzone Ein ganzzahliger Ausdruck, der die Breite der Druckzonen festlegt. Amiga Basic stellt die Druckzonenbreite mit 15 ein, wenn nichts anderes angegeben wird. Druckzonen entsprechen Tabulatorstops und werden durch ein Komma als Trennzeichen in einer PRINT– oder LPRINT–Anweisung angesprungen.

Dateinr Ein ganzzahliger Ausdruck, der die Nummer der Datei spezifiziert, die für ein Ausgabegerät eröffnet wurde.

Gerät Ein Zeichenkettenausdruck, der das Ausgabegrät spezifiziert. Folgende Geräte können angegeben werden:

SCRN:	Bildschirm
LPT1:	Drucker
COM1:	Datenfernübertragungsanschluß

Je nach Ausgabegerät kann eine der oben angegebene Schreibweisen für die WIDTH–Anweisung verwendet werden:

Syntax 1: Wird der Zusatz LPRINT bei der WIDTH–Anweisung weggelassen, so wird die Zeilenbreite für Bildschirmausgabe gesetzt. Bei gewähltem Proportionalzeichensatz können die Zeilenlängen allerdings unterschiedlich sein, weil die Zeichen unterschiedlich breit sind.

Syntax 2: Wird eine Dateinummer angegeben, so wird die Ausgabebreite für diese Datei sofort neu gesetzt, wenn sie eröffnet ist.

Syntax 3: Wird ein Gerät angegeben, so wird die neue Ausgabebreite zunächst gespeichert und wird erst wirksam, wenn für die spezifizierte Einheit eine Datei mit OPEN eröffnet wird. Die neue Breite bleibt solange wirksam, wie die Datei eröffnet ist. Ist die Datei bereits vorher eröffnet worden, ändert sich die Ausgabebreite nicht. Die Anweisungen LPRINT, LLIST und LIST,"LPT1:" benötigen kein OPEN und werden deshalb von dieser Anweisung nicht beeinflußt.

Beispiel:

```
WIDTH "LPT1:",60
OPEN "LPT1:" FOR OUTPUT AS 3
    .
    .
    .
WIDTH#3,80
```

Für den Drucker LPT1: wird eine Druckbreite von 60 Zeichen definiert, die durch die OPEN–Anweisung aktiviert und später auf 80 Zeichen geändert wird.

Anmerkungen:

- Parameterangaben außerhalb der definierten Bereiche ergeben die Fehlermeldung

  ```
  Illegal function call
  ```

  (unerlaubter Funktionsaufruf).

- Auf die Tastatur (KYBD:) hat die WIDTH–Anweisung keinen Einfluß.

- Für den Drucker hat der Interpreter eine Breite von 80 Zeichen voreingestellt.

- Bei der Änderung der Ausgabebreite für Drucker sind die physikalischen Gegebenheiten des angeschlossenen Drukkers (max. Druckbreite, engere Schrift etc.) zu berücksichtigen.

- Eine Breite von 255 löst die Zeilenstruktur auf. Dieser Wert ist für die Datenfernübertragung und Bildschirm (COM1: und SCRN:) voreingestellt.

- Eine Änderung der Ausgabebreite bei Datenfernübertragungsdateien ändert nicht die Pufferlängen.

- Nach dem Senden der angegebenen Zahl von Zeichen wird ein Wagenrücklaufcode übertragen.

- Siehe auch POS– und LPOS–Funktionen sowie PRINT– und LPRINT–Anweisungen.

WINDOW–Anweisung

Formate:

Syntax 1: **WINDOW** ***Kennung*** [,[***Titel***] [,[(***x1,y1***)–(***x2,y2***)] [,[***Typ***] [,***Schirm***]]]]
Syntax 2: **WINDOW OUTPUT** ***Kennung***
Syntax 3: **WINDOW CLOSE** ***Kennung***

Bedeutung: Ein Ausgabefenster wird definiert, aktualisiert oder geschlossen.

Syntax 1: Diese Syntax bewirkt zweierlei:

- Es wird ein neues Ausgabefenster erzeugt und im Vordergund des Bildschirmes angezeigt.
- Das neue Fenster wird aktiviert und aktualisiert. D.H., nachfolgende Anweisungen wie PRINT, CIRCLE, LINE, PUT, GET, PSET usw. wirken auf dieses Fenster.

Syntax 2: Das bezeichnete Fenster wird aktiviert und aktualisiert, **nicht** aber in den Bildschirmvordergund gebracht. Grafische Anweisungen können so im Hintergrund wirken.

Syntax 3: Das bezeichnete Fenster wird vom Bildschirm gelöscht. Wird das aktuelle Fenster gelöscht, so wird das davor zuletzt aktualisierte Fenster zum aktuellen Fenster.

Kennung Ein numerischer Ausdruck, dessen ganzzahliger Wert größer als 0 sein muß, und der bei mehreren Fenstern das gewünschte Fenster bezeichnet. Amiga Basic weist seinem Ausgabefenster beim Start die Kennung 1 zu, so daß Sie für neue Fenster eine Kennung von 2 oder größer verwenden sollten.

Titel Ein Zeichenkettenausdruck, dessen Wert als Titel verwendet wird, und der in der Titel–Leiste des Fensters angezeigt wird. Im Amiga Basic–Ausgabefenster (WINDOW 1) wird entweder BASIC angezeigt oder der Name des aktuellen Programms.

Typ Ein numerischer Ausdruck, dessen ganzzahliger Wert die Art und Weise festlegt, mit der das Fenster mit Hilfe der Maus manipuliert werden kann. ***Typ*** kann einer der folgenden Werte oder eine Summe aus diesen sein:

Wert	Bedeutung
1	Die Fenstergröße kann durch Ziehen des Größensymbols in der rechten unteren Fensterrand–Ecke mit der Maus verändert werden.
2	Das Fenster kann durch Ziehen der Titel–Leiste mit der Maus verschoben werden.
4	Das Fenster kann durch Wählen des Hintergrund– oder Vordergrund–Symbols in der rechten oberen Fensterrahmen–Ecke mit Hilfe der Maus in den Hintergrund oder Vordergund des aktuellen Bildschirms gestellt werden.

Wert	Bedeutung
8	Das Fenster kann durch Wählen des Schließ–Symbols in der linken oberen Ecke des Fensterrahmens mit der Maus geschlossen werden.
16	Der Fensterinhalt wird wieder angezeigt, wenn das Fenster zwischenzeitlich von einem anderen Fenster überlagert wurde. Amiga Basic reserviert ausreichend Speicher, um den Fensterinhalt zwischenzuspeichern.

Mehrere der angegebenen Funktionen erlauben Sie, indem Sie die zugehörigen Werte addieren. Ein Wert von 3 erlaubt z.B. durch Ziehen des Größensymbols oder der Titelleiste eine Vergrößerung oder Verkleinerung sowie die Verschiebung des Fensters auf dem Bildschirm. Jeder ganzzahlige Wert zwischen 0 und 31 ist erlaubt.

Achtung: Wenn Sie für ***Typ*** einen Wert von 17 (1 und 16) angeben, reserviert Amiga Basic ausreichend Speicher, um ein Fenster in der Größe des gesamten Bildschirmes zu speichern. In allen anderen Fällen wird nur für die angegebene Fenstergröße benötigte Speichergröße reserviert. Ein ***Typ***–Wert von 17 verbraucht also einen sehr großen Speicherbereich. Bei speicher-

kritischen Programmen sollten Sie also diese Typ–Wertekombination vermeiden.

x1,y1 Die zum aktuellen Bildschirm relativen grafischen Koordinaten der linken oberen Ecke des Fensters.

x2,y2 Die zum aktuellen Bildschirm relativen grafischen Koordinaten der rechten unteren Ecke des Fensters.

Wenn keine Koordinaten angegeben werden, erscheint das Fenster an den für dieses voreingestellten Koordinaten. Die Ausgangs–Voreinstellungen sind die Koordinaten für den gesamten Bildschirm.

Schirm Ein numerischer Ausdruck, dessen ganzzahliger Wert zwischen 1 und 4 liegen muß, und der sich auf die in einer vorausgegangenen SCREEN–Anweisung (s. dort) vergebene Kennung bezieht. Voreingestellt ist hier ein Wert von 1 (Workbench–Bildschirm).

Beispiel:

```
WINDOW 1,"Linien",(10,10)-(270,70),15
WINDOW 2,"Vielecke",(310,10)-(580,70),15
WINDOW 3,"Kreise",(10,95)-(270,170),15
WINDOW OUTPUT 1
```

Hier überdeckt das Fenster 1 ("Linien") das Standard–Ausgabefenster von Amiga Basic.

WINDOW–Funktion

Format: ***v*=WINDOW(*n*)**

Bedeutung: Liefert Informationen über das aktuelle Ausgabefenster.

n Ein numerischer Ausdruck, dessen ganzzahliger Wert zwischen 0 und 8 liegen muß und der die Art der gefragten Information festlegt:

n	übergebene Information
0	Die Kennung des **gewählten** Ausgabefensters. Dies ist das Fenster, dessen Titel **nicht** in Geisterschrift angezeigt wird.
1	Die Kennung des **aktuellen** Ausgabefensters. Dies ist das Fenster, in dem die PRINT– oder die grafischen Anweisungen wirken.
2	Die Breite des aktuellen Ausgabefensters in Bildpunkten.
3	Die Höhe des aktuellen Ausgabefensters in Bildpunkten.
4	Die x–Koordinate im aktuellen Ausgabefenster, bei der das nächste Zeichen ausgegeben wird.
5	Die y–Koordinate im aktuellen Ausgabefenster, bei der das nächste Zeichen ausgegeben wird.
6	Die Maximalzahl der für das aktuelle Ausgabefenster erlaubten Farben.
7	Ein Adreßzeiger auf den INTUITION WINDOW–Datensatz (s. Amiga Intuition–Handbuch) für das aktuelle Ausgabefenster.
8	Ein Adreßzeiger auf den RASTPORT– Datensatz (s. Amiga Intuition–Handbuch) für das aktuelle Ausgabefenster.

Beispiel: Siehe OBJECT.SHAPE–Anweisung.

Anmerkungen:

- Speziell, wenn mit mehreren verschiedenen Fenstern gearbeitet wird, ist es oft erforderlich, Informationen über ein bestimmtes Fenster einzuholen, um auf besondere Situationen reagieren zu können. Für diesen Zweck ist die WINDOW–Funktion hilfreich.

WRITE–Anweisung

Format: **WRITE** [***Liste von Ausdr***]

Bedeutung: Die Werte der Ausdrücke in der Liste werden auf dem Bildschirm ausgegeben.

Liste von Ausdr Eine beliebige Liste von numerischen und/oder Zeichenkettenausdrücken, deren Werte auf dem Bildschirm angezeigt werden sollen. Die einzelnen Ausdrücke müssen durch Komma (,) oder Semikolon (;) voneinander getrennt sein. Die Liste muß zusammen mit dem Schlüsselwort WRITE in einer Programmzeile stehen. Im Gegensatz zur PRINT–Anweisung (s. dort) gibt die WRITE–Anweisung Kommata (,) zwischen den einzelnen Werten aus und kleidet Zeichenketten in Anführungsstriche (") ein. Vor positiven numerischen Werten wird keine Leerstelle ausgegeben. Als Abschluß der Liste von Ausdrücken gibt die WRITE–Anweisung einen Wagenrücklauf/Zeilenvorschub aus. Eine WRITE–Anweisung ohne Parameter erzeugt eine Leerzeile auf dem Bildschirm.

Beispiel:

```
A=2:B=3:S$="Ergebnisse: "
WRITE S$,A+B,A*B,A↑B
```

ergibt im Ausgabefenster

```
"Ergebnisse: "5,6,8
```

Vergleiche auch Beispiel 1 bei der PRINT–Anweisung.

WRITE #–Anweisung

Format: **WRITE #*Dateinr,Liste von Ausdr***

Bedeutung: Die Werte der Ausdrücke in der Liste werden in eine sequentielle Datei ausgegeben.

Dateinr Ein ganzzahliger Ausdruck, dessen Wert die Nummer angibt, unter der die sequentielle Ausgabedatei eröffnet wurde.

Alle weiteren Spezifikationen entsprechen den bei der WRITE-Anweisung (s. dort) beschriebenen. Die Ausgabe in die sequentielle Datei erfolgt analog zur Ausgabe auf den Bildschirm, so daß auch hier auf die WRITE–Anweisung verwiesen werden kann.

Beispiel:

```
OPEN "DATEI" FOR OUTPUT AS #3
A$="Hunde-":B$="Kuchen"
WRITE #3,A$,B$
CLOSE 3
OPEN "DATEI" FOR INPUT AS #3
INPUT #3,A$,B$
PRINT A$;B$
```

ergibt im Ausgabefenster

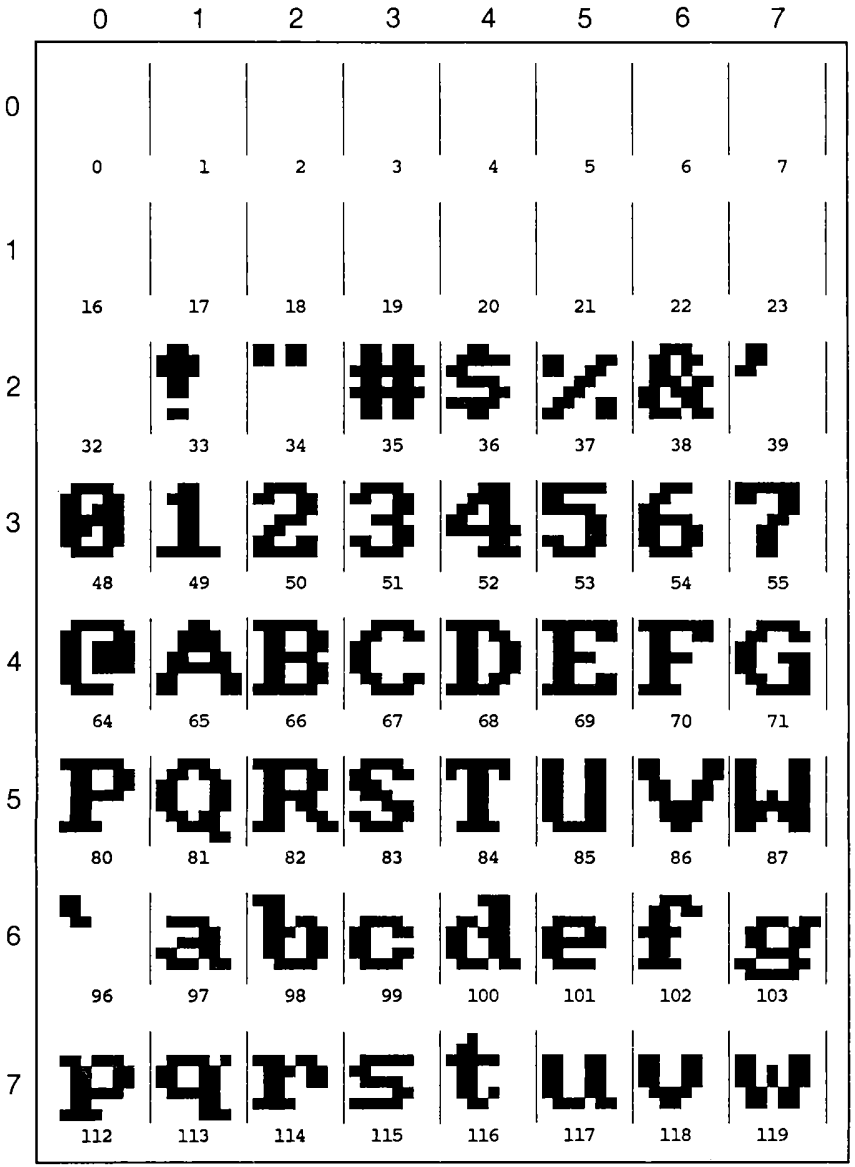
```
Hunde-Kuchen
Ok
```

Die WRITE–Anweisung schreibt das Abbild "Hunde–","Kuchen" auf Disk, wodurch bei der INPUT #–Anweisung die beiden Elemente getrennt den Variablen A$ und B$ zugewiesen werden.

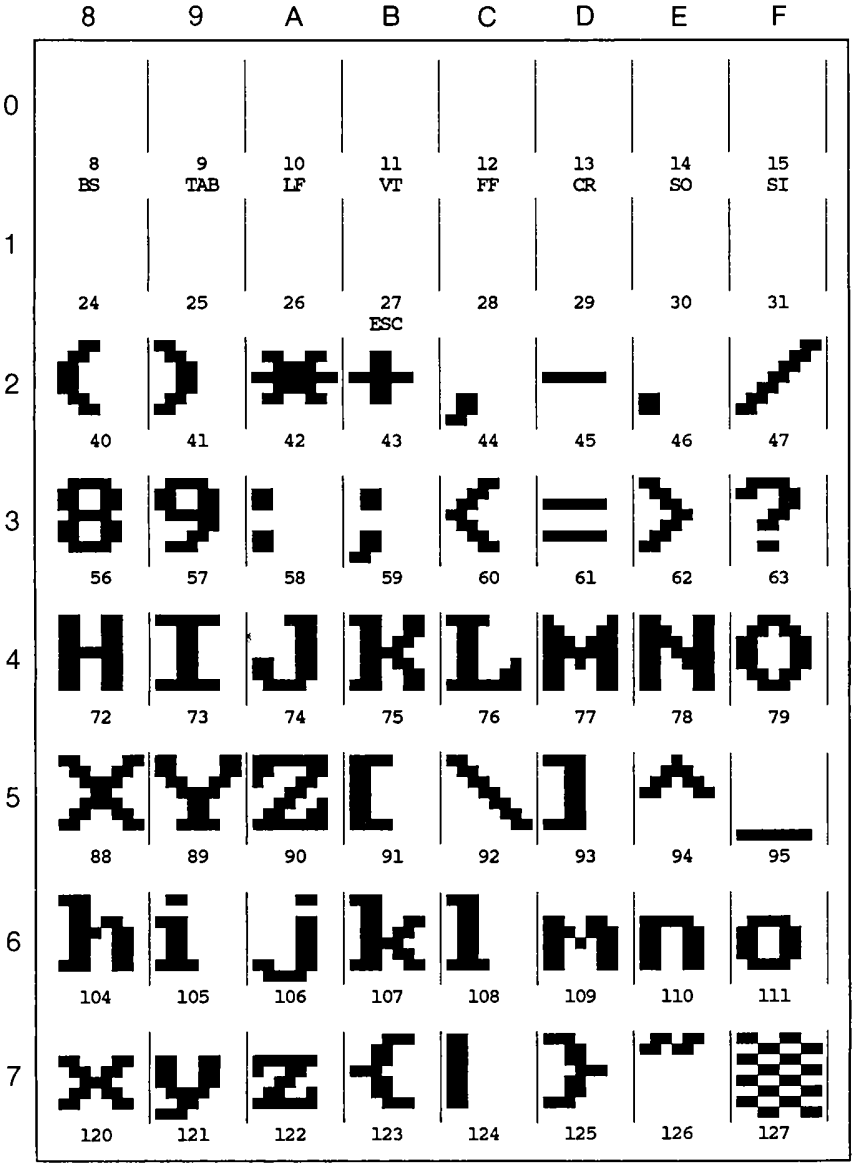
Anhang A: ASCII–Zeichencode–Tabellen

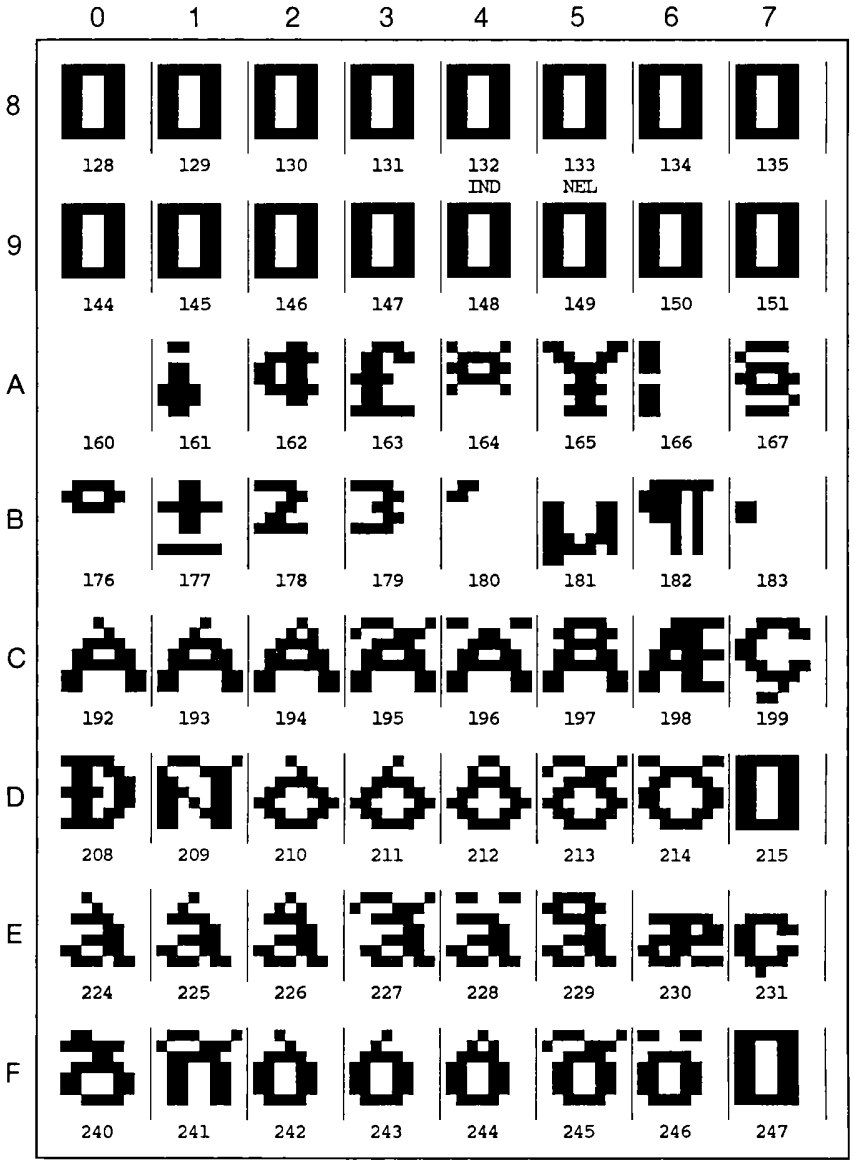
Dieser Anhang enthält die komplette ASCII- und Amiga–Zeichencodetabelle. Diese Zeichen können mit der Anweisung

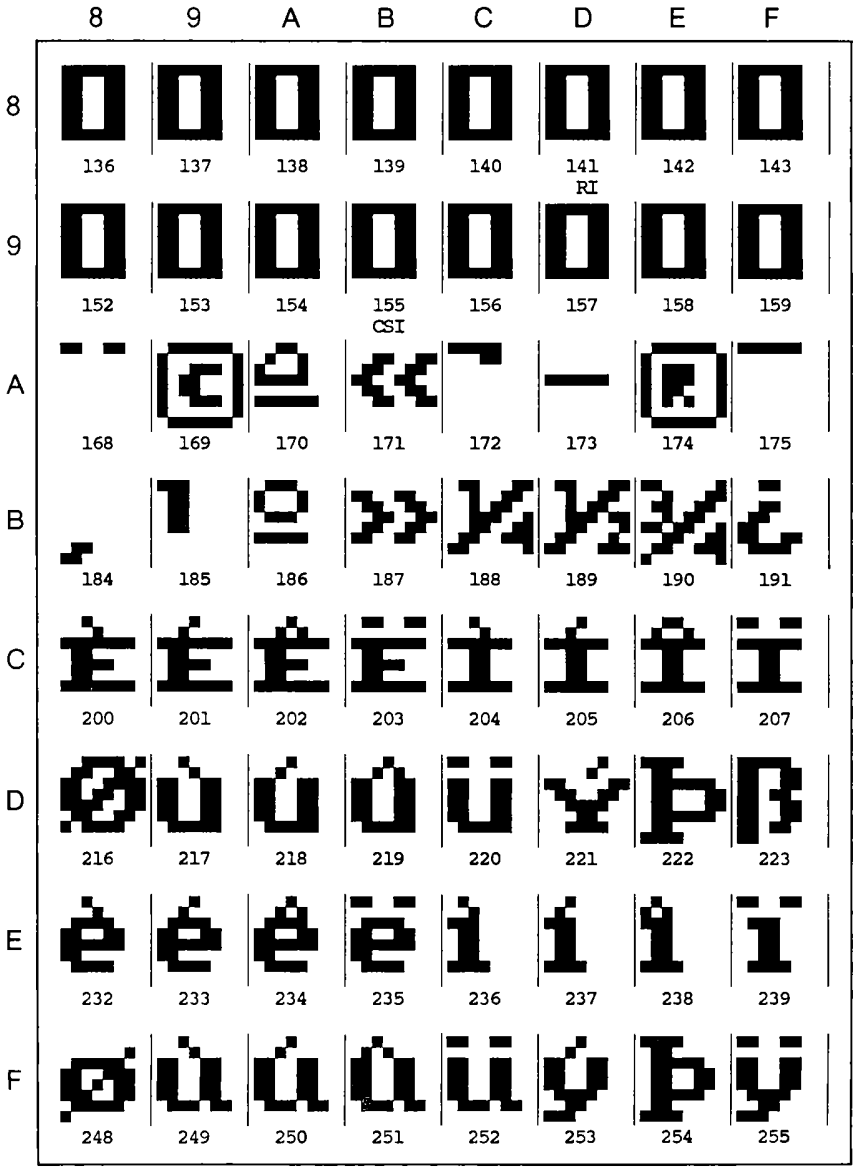
PRINT CHR$(*n*)



auf dem Bildschirm ausgegeben werden. Einige Codes ***n*** ergeben dabei kein sichtbares Zeichen, sondern haben vielmehr eine optische oder akustische Wirkung (Zeilenvorschub, Wagenrücklauf, Bildschirmlöschen, akustisches Signal etc.).




0 1 2 3 4 5 6 7
8
128 129 130 131 132 IND 133 NEL 134 135
9
144 145 146 147 148 149 150 151
A
160 161 162 163 164 165 166 167
B
176 177 178 179 180 181 182 183
C
192 193 194 195 196 197 198 199
D
208 209 210 211 212 213 214 215
E
224 225 226 227 228 229 230 231
F
240 241 242 243 244 245 246 247


8 9 A B C D E F
8
136 137 138 139 140 141 RI 142 143
9
152 153 154 155 CSI 156 157 158 159
A
168 169 170 171 172 173 174 175
B
184 185 186 187 188 189 190 191
C
200 201 202 203 204 205 206 207
D
216 217 218 219 220 221 222 223
E
232 233 234 235 236 237 238 239
F
248 249 250 251 252 253 254 255

Anhang B: Fehlercodes und Fehlermeldungen

Während der Abarbeitung eines Programms prüft Amiga Basic vor der Ausführung eines Befehls, einer Anweisung, Funktion oder Variablen deren Syntax und ggf. auch deren systematische und logische Richtigkeit. Wird ein Fehler gefunden, so wird das laufende Programm in der Regel abgebrochen, und eine entsprechende Fehlermeldung wird in einem Kommunikationsfenster am oberen Rand des Ausgabefensters angezeigt. Der Anwender muß das **Ok**–Symbolfeld anklicken, um die Meldung zu bestätigen. Zusätzlich wird bei sichtbarem List–Fenster die fehlerhafte Anweisung durch ein orangefarbenes Rechteck eingerahmt, um die Fehlersuche zu erleichtern.

Programmabbruch und Fehleranzeige können verhindert werden, wenn mit Hilfe der ON ERROR–Anweisung sowie den Systemvariablen ERR und ERL (s. dort in Kapitel 9) eine systematische Fehlerverarbeitung ins Programm eingebaut wird.

Amiga Basic unterscheidet zwischen Programmtextfehlern, Disk–Fehlern und Fehlern, die vor der Programmausführung angezeigt werden können. In den ersten beiden Fällen ordnet Amiga Basic jeder Meldung eine Fehlernummer zu. Im Kapitel B.1 sind zunächst die Programmtext– und die Disk–Fehlermeldungen in alphabetischer Reihenfolge zusammen mit ihren Fehlercodes aufgelistet. Jeder Fehlermeldung ist außerdem eine kurze Beschreibung der möglichen Fehlerursache sowie Hinweise für die Fehlerbeseitigung beigefügt. Alle Fehler, für die kein Fehlercode existiert, werden mit der Meldung

```
Unprintable error
```

(nicht druckbarer Fehler) angezeigt.

Im Kapitel B.2 folgen die Fehler, auf die Amiga Basic vor der Programmausführung prüft.

Im Kapitel B.3 sind schließlich alle Fehlermeldungen geordnet nach Fehlercode nocheinmal zusammengestellt.

B.1 Programmtext–Fehlermeldungen

Code Fehlermeldung

73 **Advanced feature** (Erweiterte BASIC–Eigenschaft)

Bedeutung: Die verwendete Amiga Basic–Interpreter–Version kann eine angegebene Anweisung nicht verarbeiten.

Abhilfe: Anweisung streichen oder umprogrammieren.

37 **Argument count mismatch**
(Argumentanzahl stimmt nicht überein)

Bedeutung: Die Anzahl der Argumente in einer CALL–Anweisung stimmt nicht mit der in der zugehörigen SUB–Anweisung überein.

54 **Bad file mode** (Falscher Dateityp)

Bedeutung: Es wird versucht, eine GET– oder PUT–Anweisung auf eine sequentielle oder nicht geöffnete Datei anzuwenden oder eine OPEN-Anweisung mit einem anderen als dem I–, O–, A–oder R–Modus auszuführen.

Abhilfe: OPEN–Anweisung überprüfen. Sicherstellen, daß beim Program–Mischen mit dem MERGE–Befehl die dazuzulandende Datei im ASCII–Format vorliegt.

64 **Bad file name** (Ungültiger Dateiname)

Bedeutung: In einer FILES–, KILL–, LOAD–, OPEN– oder SAVE–Anweisung wird ein ungültiger Dateiname verwendet.

Abhilfe: Dateiname korrigieren, so daß er den in Kapitel 5.2 beschriebenen Regeln genügt.

52 **Bad file number** (Falsche Dateinummer)

Bedeutung: Eine Anweisung oder ein Befehl bezieht sich auf eine Datei mit einer Dateinummer, unter der keine Datei eröffnet wurde, oder die außerhalb des bei der Initialisierung angegebenen Nummernbereiches liegt.

Abhilfe: Prüfen, ob die betreffende Datei korrekt eröffnet wurde oder ob die Dateispezifikation gültig ist.

Code Fehlermeldung

63 **Bad record number** (Ungültige Satznummer)

Bedeutung: Es wird versucht, in einer GET– oder PUT–Anweisung für Dateien mit wahlfreiem Zugriff eine Satznummer zu verwenden, die entweder 0 (Null) oder größer als 16777215 ist.

Abhilfe: Satznummer in der GET– oder PUT–Anweisung auf einen gültigen Wert korrigieren.

17 **Can't continue** (Programmfortsetzung nicht möglich)

Bedeutung: Ein Programm soll mit CONT fortgesetzt werden, das

- durch eine Fehlermeldung unterbrochen wurde,
- nach einer Unterbrechung verändert wurde,
- nicht existiert.

Abhilfe: Das Programm mit RUN starten oder vorher ggf. noch laden.

57 **Device I/O error** (Geräte Ein–/Ausgabefehler)

Bedeutung: Während einer Ein–/Ausgabeoperation für ein Gerät ist ein Fehler aufgetreten, der von AmigaDOS nicht behebbar ist.

Abhilfe: System rücksetzen und neu starten.

68 **Device unavailable** (Gerät nicht verfügbar)

Bedeutung Es wird versucht, eine Ein–/Ausgabeoperation für ein nicht angeschlossenes Gerät oder ein abgeschaltetes Gerät auszuführen.

Abhilfe: Gerät anschließen und Gerätezustand überprüfen oder nach Rückkehr zum Arbeitstisch (Workbench) Amiga Basic wieder aufrufen.

61 **Disk full** (Diskette oder Festplatte voll)

Bedeutung: Es steht kein Speicherplatz für weitere Daten auf der Diskette oder Festplatte im aktuellen Laufwerk mehr zur Verfügung. In diesem Fall werden alle geöffneten Dateien geschlossen.

Abhilfe: Alle nicht mehr benötigten Dateien löschen oder eine neue, formatierte Diskette verwenden.

11 **Division by zero** (Division durch Null)

Bedeutung: In einem mathematischen Ausdruck wird versucht, durch Null zu dividieren, oder das Ergebnis Null soll mit einer negativen Zahl potenziert werden. Das Ergebnis einer solchen Division ist entweder die größtmögliche darstellbare Zahl mit dem Vorzeichen der Zahl, die dividiert werden sollte, also des Dividenden, oder die positive größtmögliche darstellbare Zahl als Ergebnis einer Potenzierung.

Abhilfe: Die fehlerhafte Division oder Potenzierung korrigieren.

Code Fehlermeldung

10 **Duplicate definition** (Doppelte Definition)

Bedeutung: Es wird versucht, ein und dieselbe Feldvariable mehrmals zu dimensionieren. Dies kann folgende Ursachen haben:

- In zwei oder mehreren DIM–Anweisungen wird derselbe Feldvariablenname verwendet.
- Ein bereits mit der Standarddimensionierung von 11 Elementen dimensioniertes Feld wird noch einmal dimensioniert.
- Eine OPTION BASE–Anweisung steht nach einer DIM–Anweisung oder einer Standarddimensionierung.

Abhilfe: Programm überprüfen. OPTION BASE–Anweisung ggf. an den Programmanfang stellen.

50 **FIELD overflow** (Überlauf bei FIELD–Anweisung)

Bedeutung: Es wird versucht, in einer FIELD–Anweisung mehr Bytes zuzuordnen, als für die Satzlänge für eine Datei mit wahlfreiem Zugriff in der zugehörigen OPEN–Anweisung deklariert waren, oder es wurde während einer sequentiellen Ein–/Ausgabe mit INPUT #, PRINT # oder WRITE # in/von eine(r) Datei mit wahlfreiem Zugriff das Ende des FIELD–Puffers erreicht.

Abhilfe: OPEN– oder FIELD–Anweisung hinsichtlich der Satzlänge auf Übereinstimmung prüfen. Bei sequentieller Ein–/Ausgabe von/zu einer Datei mit wahlfreiem Zugriff darf die Länge der Daten nicht die für die Datei definierte Satzlänge überschreiten.

58 **File already exists** (Datei bereits vorhanden)

Bedeutung: Unter dem bei einem NAME–Befehl ve rwendeten Namen existiert schon eine Datei auf der spezifizierten Diskette oder Festplatte.

Abhilfe: Befehl mit anderem Namen wiederholen.

55 **File already open** (Datei bereits geöffnet)

Bedeutung: Eine bereits zum Schreiben geöffnete sequentielle Datei wird erneut zu öffnen versucht oder es wird versucht, eine geöffnete Datei zu löschen (mit KILL).

Abhilfe: OPEN–Anweisungen im Programm überprüfen. Für jede sequentielle Ausgabedatei darf nur eine OPEN–Anweisung existieren.

Code Fehlermeldung

53 **File not found** (Datei nicht gefunden)

Bedeutung: Eine in den Anweisungen FILES, KILL, LOAD, NAME oder OPEN verwendete Datei wird auf der spezifizierten Diskette oder Festplatte nicht gefunden.

Abhilfe: Disk und/oder Dateispezifikation prüfen.

26 **FOR without NEXT** (FOR ohne NEXT)

Bedeutung: Zu einer FOR–Anweisung wird vor dem physischen Programmende keine zugehörige NEXT–Anweisung gefunden.

Abhilfe: Das Programm so ändern, daß zu jeder FOR– eine NEXT–Anweisung gehört.

12 **Illegal direct** (unerlaubter Direktmodus)

Bedeutung: Es wird versucht, eine Anweisung in der Befehlsebene (Direktmodus) des Interpreters auszuführen, die nur in der Programmebene erlaubt ist (z.B. DEF FN–Anweisung).

Abhilfe: Die Anweisung darf nur in einem Programm verwendet werden.

5 **Illegal function call** (Unerlaubter Funktionsaufruf)

Bedeutung: Es wird versucht, einen Parameter außerhalb des gültigen Bereiches an eine numerische oder Zeichenketten–Funktion zu übergeben. Außerdem kann dieser Fehler in folgenden Fällen angezeigt werden:

- ein negativer oder zu großer Feldindex;
- ein ungültiges Argument für eine Funktion oder Anweisung;
- eine negative Satznummer beim Dateilesen oder –schreiben mit GET oder PUT;
- Versuch, ein geschütztes Amiga Basic–Programm zu listen oder zu verändern;
- Versuch, nichtexistente Zeilennummern zu löschen.

Abhilfe: Programm korrigieren. Ggf. bei der betreffenden Anweisung oder Funktion die Parameterversorgung überprüfen (s.a. Kapitel 9).

Code Fehlermeldung

62 **Input past end** (Eingabe nach logischem Dateiende)

Bedeutung: Es wird versucht, eine Eingabeanweisung in einem der folgenden Fälle auszuführen:

- die Datei ist leer;
- die Datei ist für Ausgabe geöffnet worden;
- es sind bereits alle Daten aus einer sequentiellen Datei gelesen.

Abhilfe: EOF–Funktion zum Testen auf Dateiende verwenden, ehe gelesen wird. Datei schließen und zum Lesen erneut öffnen. Prüfen, ob die Datei überhaupt Daten enthält.

51 **Internal error** (Interner Fehler)

Bedeutung: Bei Amiga Basic diagnostiziert AmigaDOS einen internen Fehler.

Abhilfe: Falls bei erneutem Versuch mit kopiertem Programm derselbe Fehler wieder auftritt, muß der Wartungsdienst verständigt werden.

23 **Line buffer overflow** (Zeilenpufferüberlauf)

Bedeutung: Es wird versucht, eine Zeile mit zuvielen Zeichen (>255) einzugeben.

Abhilfe: Zeile durch Aufteilen in mehrere Zeilen verkürzen. Ggf. Variablen anstelle von Konstanten verwenden.

22 **Missing operand** (Fehlender Operand)

Bedeutung: Ein Ausdruck (numerisch, logisch oder Zeichenkette) enthält einen Operator (z.B. *, AND, +- usw.), dem kein Operand folgt.

Abhilfe: Ausdruck entsprechend korrigieren.

1 **NEXT without FOR** (NEXT ohne FOR)

Bedeutung: Zu einer NEXT–Anweisung existiert keine vorangegangene FOR–Anweisung. Es ist auch möglich, daß eine hinter NEXT angegebene Variable nicht mit der bei der zugehörigen FOR–Anweisung angegebenen Laufvariablen übereinstimmt.

Abhilfe: Das Programm so ändern, daß zu jeder NEXT– eine FOR–Anweisung existiert.

Code Fehlermeldung

19 **No RESUME**
(Keine RESUME–Anweisung zur Programmfortsetzung)

Bedeutung: Das Programm ist in eine Fehlerbearbeitungsroutine verzweigt, und der Interpreter findet dort an Stelle einer RESUME–Anweisung, die die Fehlerbearbeitung beendet und das Hauptprogramm fortsetzt, nur das physische Ende des Hauptprogramms.

Abhilfe: Fehlerbearbeitungsroutine überprüfen und RESUME–Anweisung einfügen.

4 **Out of DATA** (Zu wenig Daten)

Bedeutung: Eine READ–Anweisung findet keine DATA–Daten.

Abhilfe: Für ausreichend Daten in DATA–Zeilen sorgen.

14 **Out of heap space** (Arbeitsspeicher nicht ausreichend)

Bedeutung: Der Arbeitsspeicher von Amiga Basic ist übergelaufen (z.B. wegen zu vieler geschachtelter Schleifen).

Abhilfe: Mit der CLEAR–Anweisung (s. dort in Kapitel 9) mehr Stapelspeicher zuweisen.

7 **Out of memory** (Hauptspeicher nicht ausreichend)

Bedeutung: Entweder ist das Programm zu groß oder es gibt zu viele Variablen oder Dateisteuerblöcke, so daß der Programmspeicher nicht ausreicht.

Abhilfe: Eine CLEAR–Anweisung schafft wieder Platz im Haupt– und Stapelspeicher. Ggf. muß auch das Programm verkleinert oder segmentiert werden.

6 **Overflow** (Überlauf)

Bedeutung: Das Ergebnis einer Berechnung oder eine angegebene Zahl ist zu groß, um in einem Amiga Basic–Zahlenformat dargestellt werden zu können. Bei ganzen Zahlen wird im Überlauffall das Programm abgebrochen. Sonst wird die größte bzw. kleinste darstellbare Zahl mit dem richtigen Vorzeichen übergeben und nach Ausgabe dieser Meldung das Programm fortgesetzt. Es kann also mit einer Fehlerverarbeitungsroutine nur der Überlauf bei ganzen Zahlen abgefragt werden.

Abhilfe: Entweder kleinere Zahlen angeben oder Variablen einfacher oder doppelter Genauigkeit verwenden.

Code Fehlermeldung

70 **Permission denied** (Schreiben nicht erlaubt)

Bedeutung: Es wird versucht, auf eine schreibgeschützte Diskette zu schreiben.

Abhilfe: Prüfen, ob die richtige Diskette verwendet wird. Ggf. austauschen oder Schreibschutz ausschalten (s. Amiga–Anwenderhandbuch Kapitel 3).

20 **RESUME without error** (RESUME ohne Fehlerfall)

Bedeutung: Während der Programmabarbeitung findet der Interpreter eine RESUME–Anweisung, ohne daß ein Fehlerfall vorliegt.

Abhilfe: Programm korrigieren. Wenn Fehlerbearbeitungsroutinen am Ende eines Programms angeordnet sind, fehlt davor möglicherweise eine STOP– oder END–Anweisung, so daß eine Fehlerbearbeitungsroutine ohne Fehlerfall durchlaufen wird.

3 **RETURN without GOSUB** (RETURN ohne GOSUB)

Bedeutung: Für eine RETURN–Anweisung (Ende einer Subroutine) muß eine GOSUB–Anweisung (Subroutinenaufruf) existieren.

Abhilfe: Programm korrigieren. Wenn Subroutinen am Ende eines Programms angeordnet sind, fehlt davor möglicherweise eine STOP– oder END–Anweisung, so daß eine Subroutine ohne Aufruf durchlaufen wird.

16 **String formula too complex**
Zeichenkettenausdruck zu komplex)

Bedeutung: Ein Zeichenkettenausdruck ist zu lang oder enthält z.B. zuviele geschachtelte Zeichenkettenfunktionen.

Abhilfe: Den Ausdruck in mehrere einfache Ausdrücke zerlegen.

15 **String too long** (Zeichenkette zu lang)

Bedeutung: Es wird versucht, eine Zeichenkette mit mehr als 255 Zeichen zu erzeugen.

Abhilfe: Zeichenkette in mehrere Teilketten zerlegen.

Code Fehlermeldung

36 **Subprogram already in use**
(Unterprogramm noch nicht beendet)

Bedeutung: Es wird ein Amiga Basic–Unterprogramm aufgerufen, das bereits vorher aufgerufen, aber noch nicht ordnungsgemäß beendet wurde. Rekursive, d.h. sich selbst aufrufende Unterprogramme sind nicht erlaubt.

Abhilfe: Prüfen, ob das betreffende Unterproramm ordnungsgemäß mit END SUB oder EXIT SUB beendet wird.

9 **Subscript out of range** (Index außerhalb des Bereiches)

Bedeutung: Dieser Fehler kann folgende Ursachen haben:

- Ein Feldindex wird verwendet, der außerhalb der in der zugehörigen DIM–Anweisung festgelegten Grenzen liegt;
- eine falsche Zahl von Dimensionen wird verwendet;
- ein Feldindex wird bei einer Variablen benutzt, die kein Feld ist.

Abhilfe: Entsprechende Feldvariable überprüfen. Möglicherweise wurde eine einfache Variable als Feldvariable verwendet oder bei einer Basic–Funktion wurde der Name falsch geschrieben.

2 **Syntax error** (Syntaxfehler)

Bedeutung: Der Interpreter findet in einer Programm– oder Anweisungszeile eine unkorrekte Folge von Zeichen (z.B. falsch geschriebener Befehl, ungleiche Zahl von rechten und linken Klammern, fehlerhafte Interpunktion) oder die Daten in einer DATA–Zeile stimmen im Typ nicht mit der Variablen in der READ–Anweisung überein.

Abhilfe: Die fehlerhafte Zeile muß korrigiert werden.

67 **Too many files** (Zuviele Dateien)

Bedeutung: Mit Hilfe von SAVE oder OPEN soll eine neue Datei erstellt werden, wenn

- alle Diskverzeichniseinträge bereits benutzt sind;
- eine ungültige Dateispezifikation angegeben wurde.

Abhilfe: Operation mit neu formatierter Diskette wiederholen oder Dateispezifikation korrigieren.

Code Fehlermeldung

13 **Type mismatch** (Keine Typübereinstimmung)

Bedeutung: Es wird versucht, einer Zeichenkettenvariablen einen numerischen Wert zuzuordnen oder umgekehrt, oder einer Zeichenkettenfunktion wird ein numerisches Argument gegeben oder umgekehrt oder es wird versucht, mit der SWAP–Anweisung zwei Variablen unterschiedlicher Genauigkeit miteinander zu vertauschen.

Abhilfe: Programm korrigieren; ggf. Genauigkeit der Variablen überprüfen.

38 **Undefined array** (Undefiniertes Feld)

Bedeutung: In einer SHARED–Anweisung wird ein Feld benutzt, das noch nicht dimensioniert wurde.

Abhilfe: DIM–Anweisungen überprüfen und ggf. an den Programmanfang stellen.

8 **Undefined label** (Undefinierte Marke)

Bedeutung: In einer GOTO–, GOSUB–, IF. . .THEN–Anweisung oder einem DELETE–Befehl ist eine nichtexistente Zeilennummer oder alphanumerische Sprungmarke angegeben.

Abhilfe: Programm prüfen und richtige Zeilennummer oder Sprungmarke einsetzen.

35 **Undefined subprogram** (Undefiniertes Unterprogramm)

Bedeutung: Ein nicht im Programm vorhandenes Unterprogramm wird aufgerufen.

Abhilfe: Unterprogrammnamen überprüfen. Wahrscheinlich ist ein Schreibfehler im Namen die Ursache.

18 **Undefined user function** (Undefinierte Anwenderfunktion)

Bedeutung: Eine Anwenderfunktion wird mit FN aufgerufen, ehe diese durch die DEF FN–Anweisung definiert ist.

Abhilfe: Funktion vorher mit DEF FN definieren.

Code Fehlermeldung

74 **Unknown Volume** (Unbekannte Diskette)

Bedeutung: Der Begriff 'Volume' bezieht sich auf den Namen, der beim Formatieren einer Diskette vergeben wird (s. Amiga–Anwenderhandbuch). Dieser Fehler wird angezeigt, wenn in einer Dateibezeichnung ein Disk–Name verwendet wird, der nicht mit dem Namen der eingelegten Disk übereinstimmt.

Abhilfe: Richtige Disk einlegen und Operation wiederholen.

****** **Unprintable error** (Nichtdruckbarer Fehler; kein Fehlercode!)

Bedeutung: Für die betreffende Fehlerbedingung existiert keine Fehlermeldung. Diese Meldung wird z.B. angezeigt, wenn in einer ERROR-Anweisung ein nicht definierter Fehlercode verwendet wird.

Abhilfe: Prüfen, ob für alle selbstdefinierten Fehlercodes auch Fehlerroutinen existieren.

30 **WEND without WHILE** (WEND ohne WHILE)

Bedeutung: Zu einer WEND–Anweisung existiert keine vorausgegangene WHILE–Anweisung.

Abhilfe: Das Programm so ändern, daß zu jeder WEND- eine WHILE-Anweisung gehört.

29 **WHILE without WEND** (WHILE ohne WEND)

Bedeutung: Zu einer WHILE–Anweisung wird vor Erreichen des physischen Programmendes keine zugehörige WEND–Anweisung gefunden.

Abhilfe: Das Programm so ändern, daß zu jeder WHILE– eine WEND-Anweisung gehört.

B.2 Vor der Programmausführung angezeigte Fehler

Diese Meldungen können angezeigt werden, ehe das Programm gestartet wird, da Amiga Basic beim RUN–Befehl zunächst eine formale Überprüfung des Programms vornimmt:

Block ELSE/END IF must be the first statement on the line
In einem IF–Block müssen ELSE und END IF am Anfang einer Zeile stehen.

ELSE/ELSE IF/END IF without IF
Zu ELSE, ELSE IF, oder END IF fehlt eine vorausgehende IF–Anweisung.

EXIT SUB outside of a subprogram
EXIT SUB–Anweisung außerhalb eines Unterprogramms.

FOR without NEXT
FOR–Anweisung ohne zugehörige NEXT–Anweisung

IF without END IF
In einem IF–Block fehlt die END IF–Anweisung.

Missing STATIC in SUB statement
In SUB–Anweisung fehlt der Zusatz STATIC.

NEXT without FOR
NEXT–Anweisung ohne vorausgehende FOR–Anweisung.

SHARED outside of a Subprogram
SHARED–Anweisung außerhalb eines Unterprogramms.

Statement illegal within subprogram
Unerlaubte Anweisung innerhalb eines Unterprogramms.

SUB already defined
SUB bereits deklariert

SUB without END SUB
In einem Unterprogramm fehlt die abschließende END SUB–Anweisung

Tried to declare SUB within a SUB

Versuch, SUB innerhalb einer SUB–Anweisung zu deklarieren.

WHILE without WEND

Zu einer WHILE–Anweisung fehlt die WEND–Anweisung.

WEND without WHILE

Zu einer WEND–Anweisung fehlt die vorausgehende WHILE–Anweisung.

B.3 Nach Fehlercodes geordnete Fehlermeldungen

Code	Fehlermeldung
1	NEXT without FOR
2	Syntax error
3	RETURN without GOSUB
4	Out of DATA
5	Illegal function call
6	Overflow
7	Out of memory
8	Undefined label
9	Subscript out of range
10	Duplicate definition
11	Division by zero
12	Illegal direct
13	Type mismatch
14	Out of heap space
15	String too long
16	String formula too complex
17	Can't continue
18	Undefined user function
19	No RESUME
20	RESUME without error
22	Missing operand
23	Line buffer overflow
26	FOR without NEXT
29	WHILE without WEND
30	WEND without WHILE
35	Undefined subprogram

Code	Fehlermeldung
36	Subprogram already in use
37	Argument count mismatch
38	Undefined array
50	FIELD overflow
51	Internal error
52	Bad file number
53	File not found
54	Bad file mode
55	File already open
57	Device I/O–Error
58	File already exists
61	Disk full
62	Input past end
63	Bad record number
64	Bad file name
67	Too many files
68	Device unavailable
70	Permission denied (Disk write protected)
73	Advanced feature
74	Unknown volume
**	Unprintable error

Anhang C:
Reservierte Amiga Basic-Wörter

Die im folgenden aufgeführten Wörter sind für Amiga Basic reserviert und dürfen nicht als Variablen–, Unterprogramm– oder Sprungmarkennamen verwendet werden. Andernfalls wird der Fehler `Syntax error` angezeigt.

ABS	CVI	FN	LOG
ALL	CVL	FOR	LPOS
AND	CVS	FRE	LPRINT
APPEND		FUNCTION	LSET
AREA	DATA		
AREAFILL	DATE$	GET	MENU
AS	DECLARE	GOSUB	MERGE
ASC	DEF	GOTO	MID$
ATN	DEFDBL		MKD$
	DEFINT	HEX$	MKI$
BASE	DEFLNG		MKL$
BREAK	DEFSTR	IMP	MOD
	DELETE	INKEY$	MOUSE
CALL	DIM	INPUT	
CDBL		INSTR	NAME
CHAIN	EDIT	INT	NEW
CHDIR	ELSE		NEXT
CHR$	ELSE IF	KILL	NOT
CINT	END		
CIRCLE	EOF	LBOUND	OBJECT.AX
CLEAR	EQV	LEFT$	OBJECT.AY
CLNG	ERASE	LEN	OBJECT.CLIP
CLOSE	ERL	LET	OBJECT.CLOSE
CLS	ERR	LIBRARY	OBJECT.HIT
COLLISION	ERROR	LINE	OBJECT.OFF
COLOR	EXIT	LIST	OBJECT.ON
COMMON	EXP	LLIST	OBJECT.PLANES
CONT		LOAD	OBJECT.PRIORITY
COS	FIELD	LOC	OBJECT.SHAPE
CSRLIN	FILES	LOCATE	OBJECT.START
CVD	FIX	LOF	OBJECT.STOP

Reservierte Amiga Basic–Wörter (Fortsetzung)

OBJECT.VX
OBJECT.VY
OBJECT.X
OBJECT.Y
OCT$
OFF
ON
OPEN
OPTION
OR
OUTPUT

PAINT
PALETTE
PATTERN
PEEK
PEEKL
PEEKW
POINT
POKE
POKEL
POKEW
POS
PRESET
PRINT
PSET
PUT

RANDOMIZE
READ
REM
RESET
RESTORE
RESUME
RETURN
RIGHT$
RND
RSET
RUN

SADD
SAVE
SAY
SCREEN
SCROLL
SGN
SHARED
SIN
SLEEP
SOUND
SPACE$
SPC
SQR
STATIC
STEP
STICK
STOP
STR$
STRIG
STRING$
SWAP
SUB
SYSTEM

TAB
TAN
THEN
TIME$
TIMER
TO
TRANSLATE$
TROFF
TRON

UBOUND
UCASE$
USING
USR

VA
VARPTR

WAIT
WAVE
WEND
WHILE
WIDTH
WINDOW
WRITE

XOR

Anhang D: Interne Zahlendarstellung

Amiga Basic verwendet in seinem mathematischen Teil Binär–Arithmetik. Aus diesem Grund unterscheiden sich externe und interne Darstellung von Zahlen. In den folgenden Tabellen ist die Angabe der internen Zahlendarstellung hexadezimal.

D.1 Ganze Zahlen in Amiga Basic

Ganze Zahlen werden in Amiga Basic intern in vorzeichenbehafteter 16–Bit–Zweierkomplementdarstellung verwendet:

Extern	Intern
–32768	8000
–1	FFFF
0	0000
1	0001
32767	7FFF

D.2 Gleitpunktzahlen

Bei der Binärarithmetik der Gleitpunktzahlen arbeitet Amiga Basic voreinstellungsmäßig mit einfacher Genauigkeit. Dies gilt auch für die Behandlung numerischer Variablen. Berechnungen können aber auch in doppelter Genauigkeit durchgeführt werden, benötigen dann aber erheblich mehr Zeit.

Internes Format bei einfacher Genauigkeit

32 Bit mit folgender Bedeutung von links nach rechts: 1 Vorzeichenbit
8 Bit Exponent mit Vorzeichen
24 Bit Mantisse inklusive führendem 1er–Bit

Ist das Vorzeichenbit 0, ist die Zahl positiv, ist es 1, ist die Zahl negativ.

Der Exponent ohne Vorzeichen (mit Vorzeichen –7F hex, –127 dez) ist die 2er–Potenz, mit der die Mantisse multipliziert werden muß.

Die Mantisse repräsentiert eine Zahl, die größer oder gleich 1 und kleiner als 2 ist (normalisierte Darstellung).

Positive Zahlen können bis zu, aber nicht einschließlich einem Wert von 3.4*10↑38 dargestellt werden.

Die kleinste darstellbare Zahl ist 1.18*10↑–38.

Einfachgenaue Zahlen werden mit bis zu 7.2 Stellen dargestellt.

Intern	**Extern**
1	3F800000
–1	BF800000
0	00xxxxxx
10	41200000
0.1	3DCCCCCD

Internes Format bei doppelter Genauigkeit

64 Bit mit folgender Bedeutung von links nach rechts:
1 Vorzeichenbit
11 Bit Exponent mit Vorzeichen
53 Bit Mantisse inklusive führendem 1er–Bit

Ist das Vorzeichenbit 0, ist die Zahl positiv, ist es 1, ist die Zahl negativ.

Der Exponent ohne Vorzeichen (mit Vorzeichen –3FF hex, –1023 dez) ist die 2er–Potenz, mit der die Mantisse multipliziert werden muß.

Die Mantisse repräsentiert eine Zahl, die größer oder gleich 1 und kleiner als 2 ist (normalisierte Darstellung).

Positive Zahlen können bis zu, aber nicht einschließlich einem Wert von 1.79*10↑308 dargestellt werden.

Die kleinste darstellbare Zahl ist 2.23*10↑–308.

Doppeltgenaue binäre Zahlen werden mit bis zu 15.9 Stellen dargestellt.

Extern	**Intern**
1	3FF0000000000000
–1	BFF0000000000000
0	000xxxxxxxxxxxx
10	4024000000000000
0.1	3FB9999999999999

Anhang E:
Umschreibung transzendenter Funktionen

Einige der trigonometrischen, zyklometrischen und alle Hyperbelfunktionen sind nicht Bestandteil des Amiga Basic und müssen daher wie folgt berechnet werden:

Funktion	**Umschreibung in Amiga Basic**
Logarithmus zur Basis B	LOG(X)/LOG(B)
Sekans	1/COS(X)
Cosekans	1/SIN(X)
Cotangens	1/TAN(X)
Arcussinus	ATN(X/SQR(1–X↑2))
Arcuscosinus	1.570796–ATN(X/SQR(1–X↑2))
Arcussecans	ATN(SQR(X↑2–1)) +(X<0)*3.141593
Arcuscosecans	ATN(1/SQR(X↑2–1)) +(X<0)*3.141593
Arcuscotangens	1.57096–ATN(X)
Sinus Hyperbolicus	(EXP(X)–EXP(–X))/2
Cosinus Hyperbolicus	(EXP(X)+EXP(–X))/2
Tangens Hyperbolicus	(EXP(X)–EXP(–X)) /(EXP(X)+EXP(–X))
Cotangens Hyperbolicus	(EXP(X)+EXP(–X)) /(EXP(X)–EXP(–X))
Sekans Hyperbolicus	2/(EXP(X)+EXP(–X))
Cosekans Hyperbolicus	2/(EXP(X)–EXP(–X))
Area Sinus Hyperbolicus	LOG(X+SQR(X↑2+1))
Area Cos. Hyperbolicus	LOG(X+SQR(X↑2–1))
Area Tan. Hyperbolicus	LOG((1+X)/(1–X))/2
Area Cot. Hyperbolicus	LOG((X+1)/(X–1))/2
Area Sec. Hyperbolicus	LOG((SQR(–X↑2+1)+1)/X)
Area Cosec. Hyperbolicus	LOG((SGN(X)*SQR(X↑2+1)+1)/X)

Anhang F:
Format für Maschinensprache-Bibliotheken

Dieser Anhang beschreibt, wie Amiga Basic aus Namen für Maschinensprache-Routinen den Offset in die Sprungleiste für diese Routinen im Speicher ermittelt. Diese Informationen sind für den erfahrenen Programmierer gedacht, der sich eine Bibliothek von Maschinesprache–Routinen aufbauen will, die von einem Amiga Basic–Programm aufgerufen werden können.

Da die meisten Bibliotheksroutinen in der Assembler–Sprache geschrieben werden und ihre Argumente in Registern des Mikroprozessors übergeben bekommen, müssen dem Amiga Basic–Programm die Register–Aufrufkonventionen für jede Routine bekannt sein.

Für jede von Amiga Basic mit Hilfe der LIBRARY–Anweisung (s. dort in Kapitel 9) aufrufbare Bibliothek muß eine spezielle Disk–Datei existieren, die die oben genannten Informationen enthält. Hat die Bibliothek z.B. den Namen

```
":Libs/graphics.library"
```

so erhält die spezielle Datei den Namen

```
":Libs/graphics.bmap"
```

Die Namenserweiterung `.bmap` besagt, daß es sich hier um eine spezielle Datei handelt, für die folgendes Format vereinbar ist:

Name der Routine:	*n* ASCII–Zeichen, abgeschlossen mit einem 0–Byte.
Sprungleistenoffset:	Vorzeichenbehaftete 16–Bit–Ganzzahl.

Register–Parameter: *n* Bytes abgeschlossen mit einem 0–Byte, deren Werte folgende Bedeutung haben:

1 = Parameterübergabe in Register D0
2 = Parameterübergabe in Register D1
3 = Parameterübergabe in Register D2
4 = Parameterübergabe in Register D3
5 = Parameterübergabe in Register D4
6 = Parameterübergabe in Register D5
7 = Parameterübergabe in Register D6
8 = Parameterübergabe in Register D7
9 = Parameterübergabe in Register A0
10 = Parameterübergabe in Register A1
11 = Parameterübergabe in Register A2
12 = Parameterübergabe in Register A3
13 = Parameterübergabe in Register A4

Für Routinen, die den Aufrufkonvetionen der Programmiersprache C folgen und ihre Parameter auf dem Stapel übergeben bekommen, ist der Registerparameter leer, weil Amiga Basic die Registerübergabe hier nicht benötigt.

Wenn eine Bibliothek z.B. die beiden folgenden Routinen enthält:

MoveTo(x[D0],y[D1])	mit dem Offset –24 (dezimal)
ClearRast(pRastPort [A0])	mit dem Offset –30 (dezimal),

so würde ein hexadezimaler Ausdruck des Inhaltes der .bmap–Datei dieser Bibliothek folgendermaßen aussehen:

```
4D6F7665546F00FFE80102004360656172526173740FFE20900
```

Im Unterverzeichnis "BasicDemos" auf der Extras–Diskette befindet sich das in Amiga Basic geschriebenes Hilfsprogramm **ConvertFD**, das aus einer gegebenen .fd–Datei die zugehörige .bmap–Datei erzeugt

Anhang G: Ein Beispielprogramm

Dieser Anhang enthält eine detaillierte Beschreibung des Beispielprogramms **Picture**, mit dem im Kapitel 2.1 die praktische Arbeit mit Amiga Basic beschrieben wurde. Die in eckigen Klammern den Zeilen vorangestellten Buchstaben dienen hier nur der Referenz für die Beschreibung der einzelnen Programmzeilen. Sie sind nicht Betandteil des Beispielprogramms.

```
[A]     DEFINT P-Z
[B]     DIM P(2500)
[C]     CLS
[D]     LINE(0,0)-(129,120),,BF
[E]     ASPECT = .1
[F]         WHILE ASPECT<20
[G]             CIRCLE(60,60),55,0,,,ASPECT
[H]             ASPECT = ASPECT*1.4
[I]         WEND
[J]     GET (0,0)-(127,127),P
[K]     CheckMouse:
[L]         IF MOUSE(0)=0 THEN CheckMouse
[M]         IF ABS(X-MOUSE(1)) > 2 THEN MovePicture
[N]         IF ABS(Y-MOUSE(2)) < 3 THEN CheckMouse
[O]     MovePicture:
[P]         PUT(X,Y),P
[Q]         X=MOUSE(1) Y=MOUSE(2)
[R]         PUT(X,Y),P
[S]         GOTO CheckMouse
```

Hier jetzt die Beschreibung der einzelnen Programmzeilen:

[A] Amiga Basic behandelt im folgenden alle Variablen, deren Namen mit den Buchstaben P bis Z beginnen, als Ganzzahlvariablen.

[B] Ein Feld mit einer Dimension und 2501 Elementen (0 bis 2500) wird angelegt.

[C] Das Ausgabefenster wird gelöscht.

[D] Ein Rechteck mit den Koordinaten (0,0) und (120,120) der linken oberen bzw. rechten unteren Ecke wird gezeichnet und in der voreingestellten Vordergrundfarbe (weiß) ausgemalt.

[E] Die Variable ASPECT für das Bildverhältnis wird auf den Wert 0.1 gesetzt.

[F] Die Anweisungen in den Zeilen [G] und [H] werden solange wiederholt ausgeführt, wie der Wert von ASPECT kleiner als 20 bleibt.

[G] Eine Ellipse mit dem Mittelpunkt bei (60,60) und einem Radius von 55 wird mit einem Bildverhälnis, dessen Wert gleich der Variablen ASPECT ist, in blau (voreingestellte Hintergrundfarbe) gezeichnet.

[H] Der Wert von ASPECT wird um den Faktor 1.4 vergrößert.

[I] Die Programmschleife (WHILE. . .WEND) wird verlassen, sobald ASPECT größer oder gleich 20 wird.

[J] Der definierte Fensterbereich wird als grafische, binäre Information in das Ganzzahlfeld P kopiert.

[K] Definiert den Einsprungpunkt in einen Programmteil mit dem Sprungmarkennamen CheckMouse.

[L] Wartet auf das Drücken der Auswahltaste (linke Taste) der Maus.

[M] Verzweigt zur Sprungmarke MovePicture:, wenn die Maus mindestens um 3 Bildpunkte horizontal bewegt wurde.

[N] Verzweigt zur Sprungmarke CheckMouse:, wenn die Maus mindestens um 4 Bildpunkte vertikal bewegt wurde.

[O] Definiert den Einsprungpunkt in einen Programmteil mit dem Sprungmarkennamen MovePicture.

[P] Löscht das Bild an seiner alten Position.

[Q] Setzt die Variablen X und Y auf die neuen Koordinaten der Maus.

[R] Kopiert das Bild aus dem Ganzzahlfeld P an die neue, durch X und Y gegebene Position.

[S] Verzweigt wieder zur Sprungmarke CheckMouse, wo auf Mausbewegung gewartet wird

Anhang H: Sprachausgabe mit Amiga Basic

In diesem Anhang wird beschrieben, wie Sie Lautzeichenketten erstellen, die Sie mit Hilfe der SAY–Anweisung (s. dort in Kapitel 9) in hörbare, verständliche Sprache umwandeln können. Dies wird durch den Sprach–Synthetisierer des Amiga ermöglicht. Der Sprach–Synthetisierer ist eine elektronische Einrichtung, mit der "künstliche" Computersprache erzeugt und hörbar gemacht werden kann.

In der gegenwärtigen Version des Amiga wird nur die Ausgabe angloamerikanischer Sprache von Amiga Basic komfortabel unterstützt (TRANSLATE$–Funktion), obwohl mit einigem Geschick auch andere Sprachen, darunter auch Deutsch, hörbar gemacht werden können, allerdings nur mit stark amerikanischem Akzent.

Aus diesem Grund beschäftigt sich der wesentliche Teil dieses Anhanges mit der angloamerikanischen Sprachausgabe über den Sprachsynthetisierer.

Kapitel H.10 enthält dann jedoch Hinweise, wie deutsche Sprache ausgegeben werden kann, sowie die Liste eines Amiga Basic–Programms, das mit Hilfe von zwei trickreichen Unterprogrammen Zeichenketten mit deutschem Text so aufbereitet, daß diese mit der SAY–Anweisung als deutsche Sprache ausgegeben werden können.

Um den Sprachsynthetisierer des Amiga nutzen zu können, benötigen Sie weder große phonetische oder Computer–Erfahrungen, noch Kenntnisse in einer Fremdsprache. Vielmehr benötigen Sie ein gutes Wörterbuch wie z.B. *Webster's Third International* für Englisch oder den *Duden* für Deutsch, um im Zweifelsfall die richtige Aussprache oder Betonung von Wörtern nachschlagen zu können. Das Faszinierende am Schreiben phonetischer Sprache ist, daß Sie nicht die korrekte Rechtschreibung eines Wortes, sondern nur seine korrekte Aussprache kennen müssen.

Sie können speziell englische Sprache so niederschreiben, wie Sie sie sprechen würden.

Der Sprachsynthetisierer bearbeitet sprachliche Ausdrücke auf dem Satz–Niveau. Auch wenn Sie nur ein einzelnes Wort ausgeben wollen, wird dieses als kompletter Satz aufgefaßt. Aus diesem Grunde erwartet der Synthetisierer am Ende eines jeden Satzes entweder einen Punkt (.) oder ein Fragezeichen (?). Befindet sich am Ende der auszugebenden Zeichenkette keines dieser beiden Zeichen, so fügt der Synthetisierer automatisch einen Punkt (.) an. Der Punkt, der bei fast allen Sprachausdrücken verwendet wird, resultiert in einem Abfall der Stimmhöhe am Satzende.

Das Fragezeichen, das nur am Ende von Ja–/Nein–Fragen verwendet wird, resultiert auch in einem Abfall der Stimmhöhe am Satzende. Die Frage "Do you enjoy using your Amiga?" (Macht Ihnen die Arbeit mit Ihrem Amiga Spaß ?) benutzt ein Fragezeichen am Ende, weil die Antwort ja oder nein lautet. Andererseit sollte die Frage "What is your favorite color" (Was ist Ihre Lieblingsfarbe) mit einem Punkt abgeschlossen werden. Der Synthetisierer unterscheidet noch andere Interprunktionsformen, wie später noch beschrieben wird.

H.1 Phonetisches Buchstabieren

Ausdrücke werden phonetisch gewöhnlich mit Hilfe eines Ton–Alphabets (Internationales Phonetisches Alphabet) geschrieben. Dieses phonetische Alphabet findet sich im Vorspann jedes besseren Wörterbuches. Da diese Symbole nicht leicht auswendig zu lernen sind und auch nicht auf den normalen Computer–Tastaturen verfügbar sind, hat die Organisation *Advanced Research Projects Agency (ARPA)* ein spezielles Alphabet, genannt *Arpabet*, entwickelt, mit dem jedes phonetische Symbol durch einen oder zwei Großbuchstaben des lateinischen Alphabets dargestellt werden kann.

Zur Ausgabe phonetischer Töne verwendet der Synthetisierer eine erweiterte Version des *Arpabets*.

Ein Laut, auch als **Phonem** bezeichnet, ist ein sprachlicher Grundton, also die kleinste hörbare Spracheinheit. So wird die gesprochene Sprache zunächst in Sätze, diese in Wörter, Wörter in Silben und Silben in Phoneme oder Laute unterteilt. Beispielsweise besteht das Wort **cat** (Katze) aus drei Buchstaben und geichzeitig auch aus drei Lauten. Wenn Sie jetzt einen Blick auf die Phonem–Tabelle im Kapitel H.11 werfen, sehen Sie, daß das Wort **cat** aus den drei Phonemen K, AE und T gebildet und als **KAET** dargestellt werden kann. Das Wort **cent** wird aus den Phonemen S, EH, N und T zu SEHNT zusammengesetzt. Beachten Sie hier, daß, obwohl beide Wörter mit dem Buchstaben **c** beginnen, im ersteren Fall das Phonem **K** gesetzt wird, da hier das c wie k ausgesprochen wird. Ein C–Phonem gibt es nicht; das phonetische Buchstabieren basiert nämlich auf dem Konzept:

Buchstabiere ein Wort, wie es sich ausgesprochen anhört, n i c h t wie es geschrieben wird

H.2 Wahl des richtigen Vokals

Wie geschriebene Wörter in Buchstaben zerfallen, zerfallen Phoneme in Vokale (Selbstlaute) und Konsonanten (Mitlaute). Ein Vokal ist ein kontinuierlicher Ton, der durch Stimmbandvibration bei gleichzeitigem Luftaustritt durch den Mund (**nicht** durch die Nase) gebildet wird. Alle Vokale benutzen für die Phonem–Darstellung einen Zweibuchstaben–Code. Ein Konsonant dagegen ist jeder andere Laut, der durch Luft–Zischen (s oder th) oder Luftstromunterbrechungen mit den Lippen oder der Zungenspitze (b oder t) gebildet wird. Konsonanten benutzen für die Phonemdarstellung einen Ein– oder Zweibuchstaben–Code.

Im Schrift–Englisch werden, wie im Deutschen auch, die fünf Vokale a, e, i, o, und u verwendet. Andererseits kommen im gesprochenen Englisch mehr als 15 Vokale vor. Der Sprachsynthetisierer des Amiga berücksichtigt die meisten von ihnen. Um den richtigen Vokal aus der Phonemtabelle zu wählen, muß man ihn zunächst selbst hören. Deshalb sollte man sich das betreffende Wort laut vorsprechen und ggf. dabei den gesuchten Vokal dehnend hervorheben. Dann vergleichen Sie den von Ihnen gebildeten Ton mit den Vokaltönen in den Beispielen rechts neben den Phonem–Codes in der Phonem–Tabelle. So klingt z.B. das **a** im Wort **apple** genau so, wie das **a** im Wort **cat** und nicht etwa so, wie das **a** in den Wörtern **Amiga, talk** oder **made**. Beachten Sie auch, daß einige der Beispielwörter keinen der Buchstaben enthalten, aus denen der zugehörige Phonem–Code gebildet wird, so z.B. der Code AA für den Vokal im Wort **hot**.

- Hier ist eine Anmerkung für all die Anwender angebracht, die mit dem Sprachsynthetisierer des Amiga deutsche Sprache erzeugen wollen. Bei den meisten Phonem–Codes in den Tabellen in Kapitel H.11 sind auch deutsche Beispiele angegeben. Bei einigen ist dies jedoch nicht möglich, wie z.B. beim th des Englischen. In einer späteren Version des Amiga wird auch der Sprachsynthetisierer für deutsche Sprachausgabe vorbereitet werden.

Vokale fallen in zwei Kategorien: jene, die während ihrer Dauer denselben Ton beibehalten und jene, die ihn über die Dauer ändern. Letztere werden auch als Doppelvokale (Diphthongs) bezeichnet. Sie errinnern sicher aus Ihrer Schulzeit, daß Vokale kurz oder lang klingen können. Doppelvokale klingen lang, sind aber darüberhinaus ziemlich komplex. Sprechen Sie sich z.B. das Wort **made** laut und sehr langsam vor. Beachten Sie, daß das **a** zunächst wie das **e** im Wort **bet** beginnt aber dann eher wie **e** in **beet** endet. Das **a** ist deshalb in diesem Wort ein Doppellaut, dessen Phonem–Code durch EY dargestellt wird.

Bei anderen Sprachsynthetisierern müssen Sie die Klangänderungen in Doppelvokalen als separate Phoneme angeben. Beim Amiga nimmt ihnen der Sprachsynthetisierer die Zusammensetzung der Klangänderung bei Doppelvokalen ab.

H.3 Wahl des richtigen Konsonanten

Phonetiker unterteilen die Konsonanten in mehrere Kategorien, von denen die meisten jedoch für unsere Betrachtungen keine Relevanz haben. Um den richtigen Konsonanten zu setzen, brauchen Sie nur darauf zu achten, ob er stimmhaft oder stimmlos ist. Ein stimmhafter Konsonant wird mit Stimmbandvibration, ein stimmloser ohne gebildet. Schrift–Englisch benutzt manchmal dieselbe Buchstabenkombination, um beide Fälle darzustellen. Vergleichen Sie z.B. nur den Klang von **th** in den Wörtern **thin** und **then**. Beachten Sie, daß der **th**–Klang in **thin** durch Luftströmung zwischen Zungenspitze und oberen Schneidezähnen gebildet wird; bei **then** schwingen dabei zusätzlich die Stimmbänder mit. Der Phonem–Code für das stimmhafte **th** ist DH, für das stimmlose TH. Phonetisch wird das Wort **thin** als THIHN, das Wort **then** als DHEHN geschrieben.

Zwei Laute, die besonders häufig eine Fehlerquelle in der Sprachverständlichkeit darstellen, sind das stimmhafte und das stimmlose **s**. Die phonetische Codierung hierfür ist S bzw Z. Beispielsweise endet das Wort **bats** mit dem Phonem S während **suds** mit Z endet. Sprechen Sie sich auch hier das Wort laut vor, um herauszufinden, ob es sich um ein stimmhaftes oder stimmloses **s** handelt.

Eine weitere Fehlerquelle in der Sprachverständlichkeit ist der **r**–Laut. Das Alphabet des Sprachsynthetisierers enthält für **r** zwei unterschiedliche Phoneme: R bei den Konsonanten und ER bei den Vokalen. Wenn der **r**–Laut der einzige Laut in der Silbe ist, verwenden Sie ER. Beispiele hierfür sind die Wörter **absurd, computer** und **flirt**. Wenn dagegen der **r**–Laut einem anderen Vokal vorangestellt ist oder folgt, ist R zu verwenden. Beispiele hierfür sind die Wörter **car, write** oder **craft**.

H.3 Die Verwendung von Sprachkontraktionen und speziellen Symbolen

Viele Phonem–Kombinationen in gesprochenen englischen Wörtern enstehen durch nachlässige Aussprache. So wird zum Beispiel in dem Wort **connector** das erste **o** meistens fast vollständig verschluckt, so daß anstelle des AA–Phonem-Codes der AX–Code verwendet wird. Da im Sprach–Englischen Vokale häufig nachlässig ausgesprochen werden, treten die Phonem–Codes AX und IX wiederholt vor l, m oder n auf.

Der Sprachsynthetisierer stellt für die Eingabe solcher Phonem–Kombinationen Abkürzungen zur Verfügung. Statt das Wort **personal** als PERSIXNAXL zu "buchstabieren", verwendet der Synthetisierer PERSINUL. Das Wort **anomaly** wird UNAAMULIY anstatt AXNAAMAXLIY "buchstabiert" und das Wort **combination** wird zu KAAMBINEYSHIN anstelle von KAAMBIXNEYSHIXN. Um entscheiden zu können, ob man für die Vokal–Vernachlässigung besser AX oder IX verwendet, probieren Sie beide aus und entscheiden, welche besser klingt.

Der Sprachsynthetisierer verwendet außerdem intern spezielle Symbole, die er gelegentlich in Ihren eingegebenen Satz einfügt, oder Teile des Satzes durch solche Symbole ersetzt.

Einige dieser Symbole können Sie direkt eingeben. Das wohl nützlichste ist Q (Stimmritzen–Stop), Hier wird die Luftströmung durch die Stimmritze im Kehlkopf abrupt unterbrochen. Ein Beispiel ist das Wort **Atlantic**, bei dem der Stop zwischen den Buchstaben **t** und **l** liegt. Der Synthetisierer kennt bereits diesen Stop, so daß Sie ihn nicht extra anzugeben brauchen. Sie können aber den Phonem–Code Q dort einfügen, wo sich das gesprochene Wort dann besser anhört.

H.5 Betonung und Tonfall

Nachdem Sie jetzt gelernt haben, wie Sie dem Synthetisierer das, **was** er "sagen" soll, mitteilen müssen, sollen Sie jetzt lernen, ihm mitzuteilen, **wie** er es sagen soll.

Betonungszeichen und Tonfall dienen dazu, die Bedeutung eines Satzes zu ändern, wichtige Wörter hervorzuheben und die Betonung in mehrsilbigen Wörtern auf die richtige Silbe zu legen. Damit wird die Verständlichkeit und Natürlichkeit der synthetischen Sprachausgabe weiter verbessert.

In den Phonem–Codeketten werden Betonungszeichen als einzelne Ziffern zwischen 1 und 9, denen ein Vokal–Phonem–Code folgt, angegeben. Obwohl Betonung und Tonfall nicht dasselbe sind, werden beide durch eine einzelne Zahl definiert. Neben anderen Wirkungen ist die Betonung die Verlängerung einer Silbe. Die Betonung ist also ein logischer Term: die Silbe wird betont oder nicht. Zur Kennzeichnung einer zu betonenden Silbe wird eine Zahl dem Vokal in der Silbe angefügt. Die Existenz dieser Zahl (nicht ihr Wert) weist den Synthetisierer an, diese Silbe zu betonen. Der Tonfall dagegen wird durch den Wert der Zahl definiert. Unter Tonfall wird hier das Stimmhöhenmuster oder die Kontur eines Ausdruckes verstanden.

Je größer der Wert der Betonungsziffer ist, um so höher liegt das Stimmhöhenpotential für den Akzent. Der Tonverlauf eines jeden Satzes besteht in einer rasch ansteigenden Tonhöhe bis zur ersten betonten Silbe im Satz, gefolgt von einer langsam abklingenden Tonhöhe durch den weiteren Satzverlauf bis zu einem raschen Abfall zur niedrigsten Tonhöhe bei der letzten Silbe. Weitere betonte Silben bewirken eine Unterbrechung im langsamen Abfall der Tonhöhe während des Satzverlaufes durch raschere Anstiege und Abfälle der Tonhöhe im Bereich jeder betonten Silbe. Der Synthetisierer arbeitet mit einer komplexen Prozedur, um einen möglichst natürlichen Tonhöhenverlauf entsprechend der von Ihnen gesetzten Betonungszeichen nachzubilden.

H.6 Die Verwendung von Betonungszeichen

Betonungszeichen werden direkt hinter den Vokal–Phonem–Codes angegeben. Beispielsweise gehört das Betonungszeichen im Wort **cat** hinter den Phonem–Code AE, also KAE5T. Im Allgemeinen gibt es für den Platz der Zahl in der Phonem–Code–Kette keine Alternative. Entweder folgt dem Vokal ein Betonungszeichen oder nicht. Der Synthetisierer prüft **nicht** auf richtig gesetzte Betonungszeichen bei Vokalen. Er zeigt nur einen Fehler an, wenn ein Betonungszeichen auf einen Konsonaten folgt. Für das Setzen von Betonungszeichen halten Sie sich an folgende Regeln:

1. Setzen Sie Betonungszeichen innerhalb von Wörtern, die eine besondere Bedeutung haben (Bedeutungs–Wörter). Solche Wörter sind Nomen (Hauptwörter), Verben (Tätigkeitswörter) und Adjektive (Eigenschaftswörter). Wörter wie **tonsils, remove** und **huge** sind Beispiele, bei denen einem Zuhörer mitgeteilt wird, worüber gesprochen wird. Andererseits haben Wörter wie **but, if, is** und **the** keine besondere Bedeutung und werden deshalb als Funktions-Wörter bezeichnet.

2. Setzen Sie immer ein Betonungszeichen auf die zu betonende(n) Silbe(n) von mehrsilbigen Wörtern, gleichgültig ob es sich dabei um Bedeutungs– oder Funktions–Wörter handelt. Beim Wort **Commodore** liegt die Betonung auf der ersten Silbe. In Phonem–Codes ausgedrückt heißt dies: KAA5MAXDOHR. Das Wort **Computer** wird auf der zweiten Silbe betont, also KUMPYUW5TER. Wenn Sie bei der Betonung eines Wortes im Zeifel sind, sehen Sie im Wörterbuch nach.

3. Werden in einem mehrsilbigen Wort mehr als eine Silbe betont, werden die Sekundärbetonungen nur mit einem Zahlenwert von 1 oder 2 markiert. Beim Wort **understand** werden beispielsweise die erste und letzte Silbe betont, und zwar die Silbe **stand** primär und die Silbe **under** sekundär; in Phonem–Codes also: AH1NDERSTAE4ND.

4. Zusammegesetzte Wörter wie **baseball** oder **software** werden als ein Wort geschrieben, aber wie zwei Wörter behandelt, wenn Betonungszeichen gesetzt werden. So wird also das Wort **lunchwagon** in Phonemen als LAH5NCH-WAE2GIN "buchstabiert". Beachten Sie daß die stärkere Betonung auf dem Wort **lunch** liegt, da in zusammengesetzten Wörtern immer das erste Wort am stärksten betont wird.

H.7 Setzen von Betonungswerten

Nachdem Sie die korrekte Aussprache und die richtige Plazierung von Betonungszeichen festgelegt haben, müssen Sie sich noch für den Betonungwert, also den Tonfall, für eine zu betonende Silbe entscheiden. Die einzelnen Bestandteile der englischen Schriftsprache werden nach folgender Tabelle mit unterschiedlichen Betonungswerten versehen:

Sprachelement	**Betonungswert**
Nomen (Hauptwörter)	5
Pronomen (Fürwörter)	3
Verben (Tätigkeitswörter)	4
Adjektive (Eigenschaftswörter)	5
Adverben (Ergänzungswörter)	7
Bestimmungswörter	7
Ausrufe	9
Artikel	0 ohne Beton.
Präpositionen (Verhältniswörter)	0
Konjunktionen (Verbindungswörter)	0
Sekundärbetonung	1, geleg. 2

Die Werte in dieser Tabelle sind nur als Vorschlag zu verstehen. Sie können in besonderen Fällen natürlich diese Werte auch erhöhen oder erniedrigen, ganz wie es die individuellen Anwendungen verlangen.

H.8 Setzen von Satzzeichen (Interpunktion)

Zusätzlich zu den beiden bereits erwähnten Satzendezeichen Punkt (.) und Fragezeichen (?) kennt der Synthetisierer noch den Gedankenstrich (–), das Komma (,) und die runden Klammern (). Das Komma wird dort gesetzt, wo es auch im Schrift–Englischen gesetzt würde und veranlaßt den Synthetisierer zu einer kleinen Pause mit leichtem Anstieg in der Stimmhöhe, um anzuzeigen, daß noch mehr folgt. Sie können aber auch mehr Kommata als in dem entsprechenden geschriebenen Satz setzen, um einzelne Satzteile akustisch noch deutlicher zu trennen.

Der Gedankenstrich hat dieselbe Wirkung wie das Komma, jedoch mit dem Unterschied, daß die Stimmhöhe nicht so deutlich angehoben wird. Sie kommen wahrscheinlich mit folgender Faustregel am besten zurecht:

Trennen Sie Ausdrücke mit Gedankenstrichen und Nebensätze mit Kommata.

Klammern haben eine besondere Bedeutung für die Betonungssteuerung des Synthetisierers. Klammern Sie Ausdrücke, bestehend aus zwei oder mehreren Bedeutungs–Wörtern wie z.B. **giant yacht**. Besonders effektiv können Klammern bei großen Ausdrücken wie **the silliest guy I ever saw** sein. Sie ermöglichen einen annähernd natürlichen Sprachhöhenverlauf.

H.9 Verbesserung der Sprachverständlichkeit

Obwohl Ihnen diese Einführung in die Sprachsynthetisierung einen möglichst reibungslosen Start in die eigene Arbeit mit diesem neuen Medium erlauben soll, bleibt doch der einzig sichere Weg zur Professionalität die Praxis. Folgen Sie deshalb den hier beschriebenen Tricks, um die Sprachverständlichkeit eines computer–gesprochenen Satzes zu verbessern.

1. Mehrsilbige Wörter sind häufig besser verständlich als einsilbige. Sagen Sie lieber **enormous** anstelle von **huge**. Das längere Wort enthält in allen Silben Informationen, so daß der Zuhörer mehrmals die Gelegenheit bekommt, das Wort richtig zu verstehen.

2. Optimieren Sie die Satzlänge. Orientieren Sie sich beim Schreiben für den Synthetisierer eher am gesprochenen als am gelesenen Text. Schreiben Sie keine Sätze, die nicht in einem Atemzug gesprochen werden können. Reduzieren Sie den Satzgehalt auf eine Idee.

3. Betonen Sie neue Begriffe deutlich, wenn Sie das erstemal gehört werden.

H.10 Umsetzung deutscher Schriftsprache

Mit den vorhandenen Phonem–Codes läßt sich englischer, genauer angloamerikanischer, Text in befriedigender Qualität wiedergeben. Die TRANSLATE$–Funktion nimmt dem Programmierer dabei die Auswahl der richtigen Phonem–Codes ab.

Um deutschsprachigen Text wiederzugeben, ist zweierlei nötig:

- eine Routine ähnlich der TRANSLATE$–Funktion, die Klartext in Phonem–Code–Zeichenketten umsetzt;
- ein Satz zusätzlicher Phonem–Codes, die die im Angloamerikanischen nicht vorhandenen Laute ergänzt.

Der zweite Punkt befindet sich gegenwärtig noch in Arbeit. Das erste Problem läßt sich jedoch mit relativ geringem Aufwand lösen.

Das im folgenden abgedruckte Programm enthält zwei Routinen, die zusammen die TRANSLATE$–Funktion für die Synthetisierung deutscher Sprache ersetzen.

Mit dem Subroutinenaufruf

```
GOSUB initdeutsch
```

(s. Programmliste) wird der Aufruf des eigentlichen Unterprogramms vorbereitet. Die Wandlung einer Textzeichenkette `text$`, die deutschen Text enthält, in eine Phonem–Code–Zeichenkette `deutsch$` erfolgt durch den Unterprogrammaufruf

```
german text$,deutsch$
```

Die Zeichenkette `deutsch$` kann dann mit Hilfe der SAY–Anweisung in verständliche Sprache umgewandelt werden.

In dem hier abgedruckten Beispiel liest das Demonstrationsprogramm den Text aus einer Diskettendatei `test.txt` und übergibt ihn aufbereitet der SAY–Anweisung.

Die beiden Routinen sind sicherlich noch nicht vollkommen. So werden z.B. keinerlei Betonungen gesetzt, wie das bei der TRANSLATE$–Funktion der Fall ist. Außerdem sind auch die Umlaute noch nicht implementiert. Diese müssen noch ausgeschrieben werden. Eine weitere Schwierigkeit besteht darin, daß im deutschen Text nicht immer feststellbar ist, wann ein Vokal kurz oder lang gesprochen werden soll. Im vorliegenden Beispiel wird er im Zweifel kurz gesprochen. Deshalb muß man in Einzelfällen durch gezielt falsche Schreibweise des Textes zur Verbesserung der Sprachverständlichkeit etwas nachhelfen. Beispiele hierfür sind **Commodohre** oder **Paries.**

```
' Beispiel fuer Anwendungen der SAY-Anweisungen
' zur deutschen Sprachausgabe:
' Sprechen des Textes aus der Datei test.txt

GOSUB initdeutsch
OPEN "I",1,"test.txt"
WHILE NOT EOF(1)
  LINE INPUT #1,text$
  german text$,deutsch$ 'Aufruf der Subroutine german
  SAY deutsch$,how%
WEND
CLOSE 1
SYSTEM
END
' Ende des Beispielprogramms

' Beginn der Subroutinen

initdeutsch: ' Deutschen Text sprechen
  nn=0:ni=0:RESTORE phoneme:a$=":"
  WHILE a$<>"end"
    READ a$,n
    IF n>0 THEN
      an=ASC(a$)
      IF an>nn THEN nn=an
      IF n>ni THEN ni=n
      FOR i=1 TO n
        READ b$,c$
      NEXT
    END IF
  WEND
```

```
  DIM te$(nn,ni),ph$(nn,ni),nt%(nn)
  RESTORE phoneme:a$=":"
  WHILE a$<>"end"
    READ a$,n
    IF n>0 THEN
      an=ASC(a$):nt%(an)=n
      FOR i=1 TO n
        READ te$(an,i),ph$(an,i)
      NEXT
    END IF
  WEND
  RESTORE phonemparam
  FOR j=0 TO 8
    READ how%(j)
  NEXT

initend:
  RETURN

subger:
  SUB german(t$,d$) STATIC
  SHARED te$(),ph$(),nt%()
  nn=UBOUND(te$,1)
  s$="":t$=UCASE$(t$)
  t$=" "+t$+" "
  FOR i=1 TO LEN(t$)
    a=ASC(MID$(t$,i,1))
    IF a>nn THEN nexti
    IF nt%(a) THEN
      FOR j=1 TO nt%(a)
        IF MID$(t$,i,LEN(te$(a,j)))=te$(a,j) THEN
          s$=s$+ph$(a,j):i=i+LEN(te$(a,j))-1:j=nt%(a)
        END IF
      NEXT j
    END IF
  nexti:
  NEXT i
  d$=s$
  END SUB
```

```
phoneme: ' Phonem-Codes fuer deutsche Sprache

   DATA " ",1," "," "
   DATA A,8,AEU,OY,AEH,AEAE,AY,AY,AI,AY,AH,AAAA,AU,AW,AE
   DATA B,1,B,B
   DATA C,8,CAE,TSAE," CH"," K",CHS,KS,CH,/C,CK,K
   DATA CE,TSEH,CI,TSIH,C,K
   DATA D,1,D,D
   DATA E,10,"EN ","IXN ","EL ","lXL ","ER ","IXR ","ES ","IXS "
   DATA EI,AY,EY,AY,EH,EH,EU,OY,EN,EHN,E,EH
   DATA F,1,F,F
   DATA G,1,G,G
   DATA H,1,H,/H
   DATA I,6,"IR ","IYR ",IA,IYAA,IO,IYOH,lE,IY,IH,IY,I,IX
   DATA J,2," J"," IHY",J,Y
   DATA K,1,K,K
   DATA L,1,L,L
   DATA M,1,M,M
   DATA N,1,N,N
   DATA O,6,OEH,ER,OE,ER,OU,UH,OI,OY,OH,OH,O,OH
   DATA P,2,PH,F,P,P
   DATA Q,1,Q,KV
   DATA R,1,R,R:REM oder RX
   DATA S,22,SCH,SH,"ST ","ST ",SB,SB,SD,SD,SF,SF,SG,SG,SH,S/H
   DATA SJ,SY,SK,SK,SL,SL,SM,SM,SN,SN,SP,SHP,SQ,SKV
   DATA SR,SR,SS,S,ST,SHT,SV,SF,SW,SV,SX,SKS,SZ,STS,S,Z
   DATA T,6,TIO,TSIYOH,TIA,TSIYAA,TZ,TS,TH,TT,TT,TT,T,DT
   DATA U,5,UEH,ER,UE,ER,UH,UW,"U ","UW ",U,UH
   DATA V,1,V,F
   DATA W,1,W,V
   DATA X,1,X,KS
   DATA Y,2,YH,IH,Y,IH
   DATA Z,1,Z,TS
   DATA end,0

phonemparam:

   DATA 65,0,120,0,22200,64,10,0,0
```

H.11 Phonem–Code–Tabellen

In den folgenden Phonem–Code–Tabellen ist überall da, wo es möglich ist, auch ein deutsches Wort als Beispiel eingefügt.

Vokale (Selbstlaute)

Code	Beispiele		Code	Beispiele	
	englisch	deutsch		englisch	deutsch
IY	b**ee**t	L**ie**d	IH	b**i**t	K**i**tt
EH	b**e**t	B**e**tt	AE	b**a**t	S**ä**cke
AA	h**o**t	R**o**tte	AH	**u**nder	Exp**a**nder
AO	t**a**lk	W**o**rt	UH	l**oo**k	Gl**u**cke
ER	b**ir**d	M**ö**rder	OH	b**or**der	**Or**dner
AX	**a**bout		IX	soli**d**	b**i**tter

AX und IX sollten niemals in betonten Silben verwendet werden.

Doppel–Vokale (Doppel–Selbstlaute)

Code	Beispiele		Code	Beispiele	
	englisch	deutsch		englisch	deutsch
EY	m**a**de		AY	h**i**de	b**ei**de
OY	b**oi**l	R**eu**se	AW	p**ow**er	B**au**er
OW	l**ow**		UW	cr**ew**	R**u**he

Konsonanten (Mitlaute)

Code	Beispiele englisch	deutsch	Code	Beispiele englisch	deutsch
R	**r**ed		L	ye**ll**ow	Ke**ll**e
W	**w**ay		Y	**y**ellow	**J**ugend
M	**m**en	**M**ann	N	me**n**	Kah**n**
NX	si**ng**	Ri**ng**	SH	ru**sh**	wa**sch**en
S	**s**ail	Ro**st**	TH	**th**in	
F	**f**ed	**F**utter	ZH	plea**s**ure	Men**sch**
Z	ha**s**	Ha**s**e	DH	**th**en	
V	**v**ery	**W**ein	J	**j**udge	
CH	**ch**eck	**Tsch**ako	/C	lo**ch**	Lo**ch**
/H	**h**ole	**H**und	P	**p**ut	**P**apier
B	**b**ut	**B**utter	T	**t**oy	**T**urm
D	**d**og	**D**ame	G	**gu**eest	**G**ast
K	**C**ommodore	**K**uchen			

Spezielle Symbole

Code	Beispiele englisch	deutsch	Bemerkungen
DX	pity	Ratte	Zugenstoßlaut
Q	kitt_en	Platt_en	Stimmritzen–Stop
QX	pause		stimmloser Vokal
RX	car	Haar	stimmloser Konsonant
LX	call	hohl	stimmloser Konsonant
UL	wie AXL		Kontraktion (s. Kap. H.3)
UM	wie AXM		Kontraktion (s. Kap. H.3)
UN	wie AXN		Kontraktion (s. Kap. H.3)
IL	wie IXL		Kontraktion (s. Kap. H.3)
IM	wie IXM		Kontraktion (s. Kap. H.3)
IN	wie IXN		Kontraktion (s. Kap. H.3)
1			Silbenbetonung im Bereich zwischen Sekundär- und emphatischer Betonung
.			
9			
.			Punkt. Satzende–Markierung
?			Fragezeichen. Satzende–Markierung
–			Gedankenstrich. Phrasen–Trennzeichen
,			Komma. Nebensatz–Trennzeichen
()			Klammern. Hauptphrasen – Trennzeichen (s. Kap. H.8)

Commodore